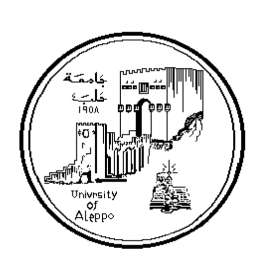


جامعـة حلـب

كلية الهندسة الكهربائية والإلكترونية

قسم هندسة الحواسيب

تطوير أداة بناء جديدة للنظم الرقمية المتكاملة باستخدام المعالجة الموزعة

Developing a new tool for integrated digital systems synthesis based on distributed processing

الدكتور يحيى نجار

قسم هندسة الحاسبات

كلية الهندسة الكهربائية والإلكترونية

جامعة حلب

الدكتور المهندس محمد أيمن نعال

قسم هندسة الحاسبات

كلية الهندسة الكهربائية والإلكترونية

جامعة حلب

المهندسة جميلة ابراهيم

طالبة دراسات عليا (ماجستير) في قسم هندسة الحاسبات

كلية الهندسة الكهربائية والإلكترونية

جامعة حلب

Aleppo University

Electrical and Electronic Faculty

Computer Engineering Department



Developing a new tool for integrated digital systems synthesis based on distributed processing

Thesis Submitted for Master Degree in Computer Engineering

Dr.Eng. Yehya Najjar

Computer Engineering Department

Electrical and Electronic Faculty

Aleppo University

Dr.Eng. Mohamad Ayman Naal

Computer Engineering Department

Electrical and Electronic Faculty

Aleppo University

By

Eng. Jamila Ebrahim

قدمت هذه الرسالة استكمالاً لمتطلبات نيل درجة الماجستير

في هندسة الحواسيب في قسم هندسة الحاسبات

من كلية الهندسة الكهربائية والإلكترونية بجامعة حلب

شهادة

نشهد بأن العمل المقدم في هذه الرسالة هو نتيجة بحث علمي قام به المرشح المهندسة جميلة ابراهيم تحت إشراف الدكتور المهندس يحيى نجار (المشرف الرئيسي) الأستاذ المساعد في قسم هندسة الحاسبات في كلية الهندسة الكهربائية والإلكترونية بجامعة حلب والدكتور محمد أيمن نعال (المشرف المشارك) المدرس في قسم هندسة الحاسبات في كلية الهندسة الكهربائية والإلكترونية بجامعة حلب وأنّ أيّة مراجع أخرى ذكرت في هذا العمل موثقة في نص الرسالة.

المرشح	**المشرف المشارك**	**المشرف الرئيسي**
المهندسة جميلة ابراهيم	**الدكتور محمد أيمن نعال**	**الدكتور يحيى نجار**

تصريح

أصرح بأن هذا العمل :

تطوير أداة بناء جديدة للنظم الرقمية المتكاملة باستخدام المعالجة الموزعة

لم يسبق أن قُبل للحصول على أي شهادة، ولا هو مقدم حاليًا

للحصول على شهادة أخرى.

المرشح

المهندسة جميلة ابراهيم

منهجية البحث

لقد اتبعت في البحث المقدم المنهجية التالية:

المرحلة الأولى: إجراء تصنيف عام للنظم الرقمية واستنتاج المعادلة العامة لهذه النظم .

المرحلة الثانية: دراسة مرحلة الترجمة من مراحل بناء الأداة المنطقية عالي المستوى.

المرحلة الثالثة: دراسة مرحلة الجدولة من مراحل بناء الأداة المنطقية عالي المستوى.

المرحلة الرابعة: مقارنة لغات البرمجة الثلاث: C#-Delphi-Java بالنسبة لمرحلتي الترجمة والجدولة.

المرحلة الخامسة: توزيع مرحلة الترجمة على شبكة خارجية.

المرحلة السادسة: توزيع مرحلة الترجمة على شبكة داخلية ومقارنة النتائج مع الشبكة الخارجية.

ولقد تم تحقيق دراستين مستقلتين خلال البحث وهما:

1. **دراسة تأثير لغة البرمجة على مرحلتي الترجمة والجدولة.**
2. **دراسة تأثير نوع الشبكة المستخدمة في التوزيع لمرحلة الترجمة.**

المنشورات

تم خلال فترة البحث نشر بحث علمي في مجلة بحوث حلب وفق الآتي:

- البحث الأول: "دراسة مقارنة وتقويم للخوارزميات المستخدمة في بناء النظم الرقمية" وذلك في العدد /66/ لعام 2009 من مجلة بحوث جامعة حلب سلسلة العلوم الهندسية علماً أن البحث ورد للمجلة بتاريخ 2008/10/29 وقبل للنشر بتاريخ 2009/1/18.

قائمة الأشكال

قائمة الجداول

الإختصارات

الإختصار	نشره	الصفحة
SOC	System-on-Chip	1
ASIC	Application Specific Integrated Circuit	2
CISC	Complex Instruction Set Computer	2
RISC	Reduced Instruction Set Computer	3
DSP	Digital Signal Processor	3
MAC	Multiply and Acumulate	3
DMA	Direct Memory Access	3
MIMD	Multiple Instruction stream, Multiple Data stream	3
IIR	Infinite Impulse Response digital filters	4
HLS	High Level Synthesis	5
LLS	Low Level Synthesis	5
DFG	Data/Control Flow Graph	6
SDFG	Scheduled Data Flow Graph	6
RTL	Register Transfer Level	7
FUs	Functional Units	8
ASIC	Application Specific IC	8
ASAP	As Soon As Possible	31
ALAP	As Late As Possible	31
VHDL	Very High Speed Hardware Description Language	52
TCP	Transmission Control Protocol	92
IP	Internet Protocol	92
ASCII	American Standard Code for Information Interchange	92
UDP	User Datagram Protocol	93
ACK	Acknowledgment	94

فهرس المحتويات

الفصل الأول

مقدمة عامة عن النظم الرقمية

1-1-أهمية البحث:[1]

تعتبر النظم الرقمية من أهم الأنظمة المستخدمة في مختلف مجالات الحياة العملية والتي تتزايد أهميتها بشكل كبير لما لها من سرعة ودقة عالية مقارنةً مع النظم التشابهية.

مع التقدم السريع في تكنولوجيا تصنيع النظم المتكاملة أصبح بإمكاننا اليوم تصنيع نظم رقمية شديدة التعقيد وعالية السرعة على رقاقة واحدة مما أدى إلى ظهور مصطلح النظام على رقاقة System-on-Chip(SoC) ، ولهذا فإن عملية تصميم النظم الرقمية أصبحت من العمليات المعقدة والحساسة والتي تتطلب الكثير من البحث والتطوير لتقليل كلفة التصميم من حيث الزمن ومساحة الرقاقة واستهلاكها وما إلى ذلك من القيود التي يمكن فرضها على النظام المصمم ولضمان جودة النظام الناتج وخلوه من الأخطاء التصميمية والتصنيعية.

في هذا السياق تعتبر كافة مراحل بناء النظم الرقمية (Digital system synthesis) من المراحل المعقدة والتي تحتاج إلى تطوير أدوات تصميم جديدة تلبي متطلبات التطور، سواءً مراحل التصميم عالية المستوى أو منخفضة المستوى، مما يجعل عملية الحصول على أداة تصميم متكاملة أمراً بالغ التعقيد.[2]

هذه الرسالة تعتبر خطوة أولى في تطوير أداة تصميم ومحاكاة للنظم الرقمية المتكاملة بجهود وطنية وبهدف كسر الحصار التكنولوجي في هذا المجال وذلك عبر اقتراح تصنيف عام لخوارزميات الجدولة المستخدمة في تصميم النظم الرقمية مع بيان مجالات استخدامها والقيود التي يمكن فرضها على النظام المصمم مع دراسة إمكانية تحويلها إلى معالجة موزعة. هذا التصنيف سيساعد على فهم طريقة التعامل مع القيود المفروضة على التصميم لكل صنف من أصناف النظم الرقمية ويؤمن دراسة مقارنة لإمكانية تطبيق مبدأ المعالجة الموزعة على مختلف أنواع الخوارزميات، ثم اقتراح خوارزمية جديدة تعتمد هذا المبدأ.[3]

خلال الصفحات التالية سنتطرق بشكل مختصر إلى النقاط التالية:

- مقدمة في تصنيف النظم الرقمية المتكاملة.
- مراحل بناء نظم الحاسبات الرقمية المتكاملة.

1-2-مقدمة في تصنيف النظم الرقمية المتكاملة:[1,4]

سنستعرض هنا تصنيفاً عاماً للنظم الرقمية المتكاملة من حيث طريقة التنفيذ بغية استخلاص العلاقة الرياضية التي تصف النظام المطلوب، وهي بهذا تنقسم إلى ثلاثة أقسام رئيسية:

1-2-1- النظم المصنعة خصيصاً للتطبيق ASIC (Application Specific Integrated Circuit):

وهي النظم المتكاملة التي تم تصميمها وتصنيعها خصيصا لتطبيق معين بحيث لا يمكن إعادة برمجتها أو تشكيلها لتطبيق آخر. هذا النوع من النظم المتكاملة يمتاز بأنه يقدم أفضل الميزات للنظام المطلوب من حيث السرعة ومساحة الرقاقة واستهلاك الطاقة ودقة الأداء وجودته وغيرها إلا أن النظام المصنع بهذه الطريقة لا يمكن إعادة تشكيله لاستعماله في تطبيق آخر.

1-2-2- نظم المعالجات الرقمية (Digital Microprocessors):

هذه النظم هي عبارة عن دارة متكاملة تحتوي على وحدة معالجة مركزية مع ملحقاتها من وحدات الدخل والخرج ووحدات التحكم والذواكر. قد أصبحت هذه النظم الأكثر شهرة والأكثر مبيعاً في العالم وذلك للأسباب الآتية:

- الكثافة العالية في تصنيع الدارات المتكاملة والتي سمحت بتنفيذ معالجات متطورة.
- التقنيات المتقدمة في التصنيع والتي سمحت بالعمل على ترددات عالية جداً.
- كثافة الإنتاج والتي أدت إلى خفض أسعارها واستعمالها في الكثير من التطبيقات العامة والخاصة.

يمكن تصنيف المعالجات الرقمية حسب بنيتها إلى:

أ- المعالجات التقليدية CISC (Complex Instruction Set Computer):

وهي المعالجات التي تعتمد على مجموعة كبيرة من التعليمات المعقدة والتي تتطور مع الزمن وتزداد تعقيداً بحسب متطلبات التوافق مع معالجات الجيل السابق. في هذا النوع من المعالجات هناك ارتباطاً وتفاعلاً بين بنية المعالج واستراتيجية البرنامج المنفذ عليه.

ب- المعالجات ذات مجموعة التعليمات المخفضة RISC (Reduced Instruction Set Computer):

والتي تحدد فيها مجموعة التعليمات الخاصة بالمعالج بشكل مبسط بهدف تخفيض عدد دورات الآلة اللازمة للتعليمة. في هذا النوع من المعالجات هناك فصل بين استراتيجية البرنامج وبنية المعالج.

ت- معالجات الإشارة الرقمية DSPs (Digital Signal Processors):[5]

وهي معالجات تأخذ بعين الاعتبار المتطلبات الخاصة بمعالجة الإشارة الرقمية وتتم دراسة بنيتها لتسهيل تنفيذ التعليمات المتعلقة بتطبيقاتها، وتتميز بما يلي:

- تتمحور تعليمات هذا النوع من المعالجات حول تعليمة رئيسية من نوع الضرب والجمع (Multiply and Acumulate MAC) بحيث تسمح بتنفيذ هاتين العمليتين في دورة آلة واحدة. وهذا النوع من العمليات نجده بكثرة في حساب السلاسل وضرب المصفوفات.
- وجود تعليمات تسمح بالقيام بأكثر من مهمة خلال دورة الآلة خاصة في مسألة التحكم بحلقات التكرار وعنونة الجداول في الذاكرة.
- تقنية الوصول المباشر إلى الذاكرة (Direct Memory Access DMA) المدمجة مع هذا النوع من المعالجات لتسهيل النقل المباشر لكميات كبيرة من المعطيات.
- ذواكر محلية خاصة بالمعالج.
- إمكانية التغيير السريع في منحى العمل.

ث- معالجات الشبكات (Multiple Instruction stream, Multiple Data stream MIMD):

والتي تسمح ببناء النظم التفرعية التي تتألف من مجموعة خلايا معالجة تتصل معاً بواسطة الرسائل. فبالإضافة إلى معالجة المعطيات تحوي هذه المعالجات على إمكانيات خاصة بالإتصال بالمعالجات الأخرى مع ملاحظة توزيع المهام الخاصة بالتطبيق بحيث نقلل من الحاجة إلى إرسال الرسائل.

1-2-3- النظم الميكروية(Microsystems):

وهي نظم كهرميكانيكية متكاملة تجمع على نفس الدارة المتكاملة الحساسات الكهركيميائية والكهرميكانيكية وعناصر التأثير الكهربائي والميكانيكي، كالمحركات الكهربائية وأذرع الدوران ونقاط استنادها ومسننات نقل الحركة، ودارات القيادة الرقمية والتشابهية الخاصة بها. هذا النوع من النظم قد شهد تطوراً ملحوظاً منذ عام 1996وأصبحت له تطبيقات هامة جداً .

1-3- مجال البحث:

إن الدراسة التي أجريت في هذه الأطروحة يمكن تطبيقها بشكل عام على أي نظام رقمي مع الأخذ بعين الاعتبار المتطلبات الخاصة لكل صنف من هذه النظم. ولصعوبة تقديم مثال تطبيقي لهذه الدراسة على كل صنف من النظم الرقمية فقد تم تطبيق الدراسة موضوع هذه الأطروحة على نوع خاص من نظم الحاسبات الرقمية وهي نظم معالجة الإشارة الرقمية (Digital Signal Processors DSPs) وبشكل خاص على المرشحات الرقمية من نوع IIR (Infinite Impulse Response digital filters) وتم اختيار هذا النوع من النظم للأسباب الآتية:[6,7,8]

- التطبيقات الهامة والمتعددة لهذه النظم في كثير من المجالات.
- التعقيد النسبي في تصميمها وتصنيعها من حيث استقرار النظام والقيود المفروضة عليه من مساحة رقاقة وسرعة استجابة واستهلاك طاقة وغيرها.
- سهولة تطوير طرق الاختبار والبناء المطبقة عليهالتشمل أنواعاً أخرى من النظم الرقمية العامة والمتخصصة (FIR and multi-rate filters, ALUs, …).

إن المعادلة العامة لهذا النوع من المرشحات هي من الشكل:

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) - \sum_{i=1}^{N} b_i y(n-i)$$

انطلاقاً من التوصيف المعطى للنظام يتم استخلاص العلاقات الرياضية للعمليات المطلوب تنفيذها على المعطيات وتصاغ على شكل معادلات دخل لأداة التصميم وذلك بالصيغة Infix. إن العلاقة التي سيتم ترجمتها وجدولتها في هذا البحث هي علاقة مشتقة من الصيغة العامة لمعادلة مرشح رقمي IIR وشكلها:[9,10,11]

$$Y=a_1*b_1+a_2*b_2+a_3*b_3+\ldots.+a_n*b_n$$

$$y(n) = \sum_{k=1}^{N} a_k * b_k$$

1-4- مراحل بناء نظم الحاسبات الرقمية المتكاملة:[12,13,14]

تعتبر عملية تصميم النظم من العمليات المعقدة والتي تتطلب يوماً بعد يوم المزيد من الأبحاث بهدف تحسينها. يمكن من حيث المبدأ تقسيم عملية التصميم إلى ثلاث مراحل هي:

1- مرحلة التوصيف (Description): وتتم باستخدام لغة عالية المستوى متخصصة في توصيف النظم العتادية (Hardware)، كما يمكن مشاركة أكثر من لغة عالية المستوى في التوصيف خاصة فيما يعرف بـ Hardware/Software co-design.

2- مرحلة البناء (Synthesis): وهي مرحلة إنشاء النظام انطلاقاً من التوصيف المعطى وصولاً إلى الدارة المتكاملة النهائية التي تؤدي الغرض المطلوب من النظام وتقسم إجمالاً إلى قسمين رئيسيين هما:

- مرحلة البناء عالي المستوى High Level Synthesis (HLS).
- مرحلة البناء منخفض المستوى Low Level Synthesis (LLS).

3- مرحلة التنفيذ (Implementation): ويمكن اعتبارها جزءاً من مرحلة البناء إلا أنها تختلف عنها في تحديد البنية الفيزيائية النهائية للنظام ويمكن الانتقال إليها من مستويات متعددة من مرحلة البناء.

وسنهتم فيما يلي بتوضيح هذه المراحل مع التركيز على مرحلة البناء عالي المستوى HLS من مراحل بناء النظم.

1-4-1- مرحلة توصيف النظام (System description):[1,13,15]

يمكن توصيف النظم بأحد الأسلوبين التاليين:

1- التوصيف السلوكي (Behavioral description) وفيه يهتم المصمم بسلوك النظام الكلي عبر علاقة المداخل بالمخارج بشكل أساسي.

2- التوصيف البنيوي (Structural description) وفيه يهتم المصمم بالبنية الهرمية للنظام مع التوصيف السلوكي أو البنيوي للعناصر الأساسية التي يتكون منها النظام.

يمكن أن يكون الوصف السلوكي أو الوصف البنيوي للنظام في هذه المرحلة مكتوباً بلغة VHDL وهي لغة التوصيف العتادي القياسية [2].

1-4-2- مرحلة بناء النظام (System Synthesis):[16]

في هذه المرحلة يتم تحويل النظام من الفكرة المجردة إلى النظام الفيزيائي النهائي المعد للعمل ضمن البيئة المحددة له، ويتم هذا البناء باستخدام أدوات تصميم متطورة ويمر عبر مرحلتين أساسيتين هما مرحلة البناء عالي المستوى ومرحلة البناء منخفض المستوى. تبدأ المرحلة الأولى بوصف النظام المراد تصميمه وتنتهي المرحلة الثانية بالدارة المتكاملة التي تمثل النظام المطلوب.

1-4-2-1- مرحلة البناء عالي المستوى (High Level Synthesis):[17,18]

تتضمن هذه المرحلة أربع خطوات رئيسية تشتمل على:

1- ترجمة الوصف الألغوريتمي للنظام (Compilation) إلى مخططات انسياب المعطيات و/أو التحكم (Data/Control Flow Graph) والذي يمثل عمليات النظام وتسلسل تنفيذها، ونرمز له اختصاراً بـ DFG.[12,16]

2- جدولة عمليات النظام (Scheduling) الممثلة في DFG في خطوات زمنية تسمى بخطوات التحكم (Control steps) ونحصل بهذا على مخطط انسياب معطيات مجدول Scheduled Data Flow Graph (SDFG).[19]

3- تخصيص مجموعة من الوحدات العملياتية (Functional Units) المتوفرة وربط عمليات النظام المجدولة بهذه الوحدات وتوضيع المسجلات (Registers) والنواخب (Multiplexers) وخطوط النقل (Buses) اللازمة (Resource allocation and Binding). وبهذا يتولد لدينا ما يسمى بممر المعطيات (Data path).

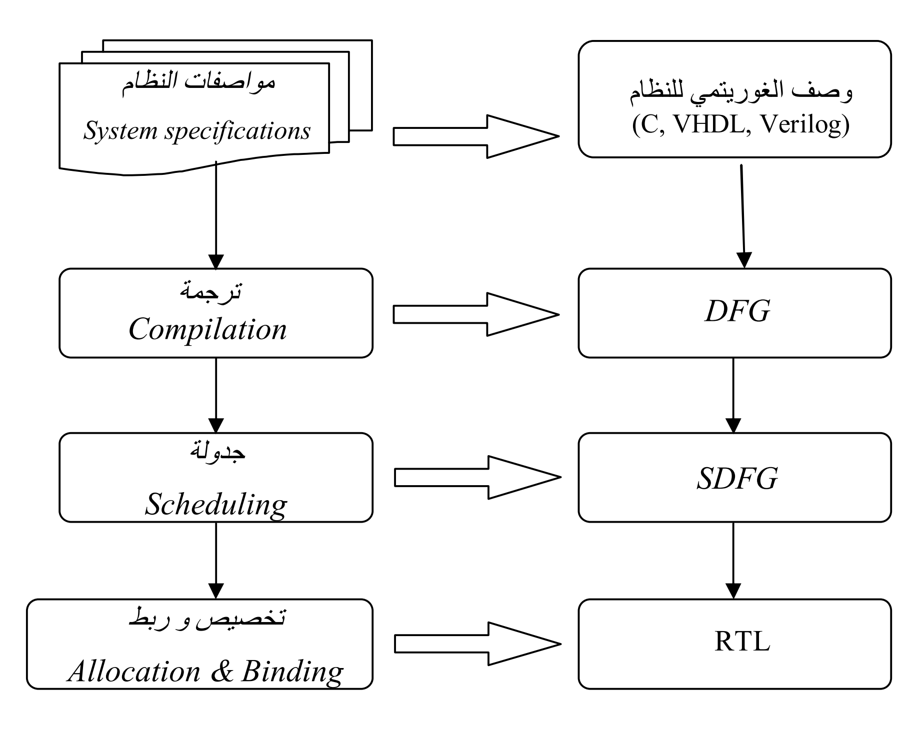
4- توليد قسم التحكم الذي سيقود مختلف أقسام النظام (Control path).

وبهذا يتم تجميع النظام في هذه المرحلة على شكل وحدات عملياتية مع المسجلات اللازمة والنواخب وخطوط النقل التي تربط مختلف أجزاء النظام بعضها ببعض إضافة إلى نظام التحكم الذي يقود عمل هذه الجملة حسب المطلوب منها. هذا المستوى يسمى مستوى النقل بين المسجلات Register Transfer Level (RTL).[10]

1-4-2-2- مرحلة البناء منخفض المستوى (Low Level Synthesis):

تتضمن هذه المرحلة توصيف الوحدات العملياتية، المستخدمة في بناء النظام على مستوى RTL، بواسطة الدارات المنطقية الممثلة لها، ثم يتم تمثيل هذه الدارات المنطقية بالترانزيستورات وهذه الترانزيستورات تمثل بدورها بدارتها المتكاملة (طبقات نصف الناقل والمعدن التي تشكل الترانزيستور). بهذا الشكل يتم توليد النظام المطلوب على شكل دارة متكاملة معدة للتصنيع على شكل ASIC. أما في حالة تنفيذ النظام على دارات مبرمجة (Microprocessor, Micro controller, FPGA) فإنه يكتفى بالمرحلة الأولى من بناء النظام ويتم إرسال النظام إلى أدوات خاصة بالدارة المبرمجة المقصودة ليتم توضيع النظام عليها.[20]

يبين الشكل(1-1) مراحل البناء عالي المستوى لنظم الحاسبات الرقمية ومستوى توصيف النظام في كل مرحلة.



الشكل(1-1) مراحل البناء عالي المستوى.

1-4-3- مرحلة تنفيذ النظام (System implementation)[10,20] :

يمكن تنفيذ النظام النهائي بأشكال مختلفة بحسب ما تفرضه بيئة العمل وقيود التصميم، منها التنفيذ البرمجي (Software systems) ومنها العتادي (Hardware systems) ومنها المشترك (Embedded systems).

1- التنفيذ البرمجي (Software) ويتم انطلاقاً من مرحلة الجدولة بتحديد عملية واحدة في كل خطوة زمنية ثم تترجم كل عملية إلى لغة الآلة الخاصة بالنظام الذي سيشغل البرنامج.

2- التنفيذ العتادي (Hardware) وذلك باستخدام وحدات عملياتية FUs (Functional Units) معدة مسبقاً، حيث يتم توضيع الوحدات العملياتية وتوصيلها انطلاقاً من وصف النظام في مستوى RTL. استخدام الدارات القابلة للبرمجة (PAL, PLA, PLD, FPGA, ...) ويتم انطلاقاً من توصيف النظام في مستوى البوابات المنطقية (Gate level) ثم يتم إسقاط هذه البوابات على الدارات القابلة للبرمجة.

3- التنفيذ كدارة متكاملة مخصصة ASIC (Application Specific IC) وفيها نحتاج إلى متابعة مراحل التصميم إلى مرحلة Layout.

في هذه الأطروحة تم التركيز على مرحلتي الترجمة والجدولة (Compilation and Scheduling) مع عرض تصنيف عام لهذه الخوارزميات وذلك باستخدام ثلاث لغات برمجية وهي (C#, Delphi, Java) بغية اختيار اللغة الأنسب لمتابعة العمل في بناء الأداة المنطقية واقتراح خوارزميات حول المعالجة الموزعة.

الفصل الثاني

مرحلة الترجمة (Compilation)

2-1- خطوات مرحلة الترجمة:[16,17,19]

- يتم أولا تحويل معادلات الدخل من الصيغة Infix إلى الصيغة Postfix كما يمكن تحويلها إلى الصيغة Prefix لتحديد أولويات العمليات وحذف الأقواس إن وجدت استعداداً لبناء مخطط انسياب المعطيات Data Flow Graph (DFG).

- يتم تحويل الصيغة Postfix لمعادلات الدخل إلى مخطط انسياب المعطيات (DFG) على شكل شجرة ثنائية (Binary tree) مؤلفة من لائحة مترابطة (Linked list) أو مصفوفة (ويفضل استخدام المصفوفات الديناميكية هنا) باستخدام البنية التالية للسجل (record) الموضحة في الشكل (2-1).

Var Left	Next Left	Operation	Next Right	Var Right

الشكل (2-1) بنية السجل في اللائحة المترابطة قبل عملية الجدولة.

- تتم قراءة الصيغة Postfix من نهايتها أي من العملية التي تعطي قيمة الخرج.

- يشار إلى نهاية هذه اللائحة المترابطة بمؤشر root وترتبط كل عملية مع العملية التالية لها عبر حقل Next.

- إذا كان ارتباط العملية بأحد متغيرات الدخل الرئيسية للمعادلة، يوضع في حقل Next المقابل لهذا الدخل القيمة Null ويوضع اسم المتغير في الحقل المقابل له (Var Right or Var Left).

- إذا كان ارتباط العملية بعملية سابقة لها فيتم إنشاء سجل للعملية الجديدة ويوضع عنوان العملية السابقة في الحقل المقابل له (Next Right or Next Left).

وبالتالي باستخدام خوارزميات الترجمة المناسبة سيكون الدخل عبارة عن معادلة رياضية والخرج عبارة عن لائحة مترابطة أو مصفوفة ثابتة أو ديناميكية تعبر عن مخطط انسياب المعطيات DFG.

2-2- الصيغ INFIX – POSTFIX – PRIFIX:[21,22,23]

اعتدنا دائما أن نعبر عن عملية جمع عنصرين A،B بالشكل A+B وهذا يسمى صيغة التدوين الوسطي أو Infix إلا أن الرياضي البولوني JAN LUKASIWICZ(1878_1956) اقترح أن نعبر عن عملية الجمع تلك بالشكل +AB وسمى هذه الصيغة بالصيغة البولونية أو PREFIX. يمكن أن نعبر أيضاً عن عملية الجمع تلك بشكل ثالث +AB وهذا يسمى Postfix أو الصيغة البولوني المعكوس، حيث تعبر البادئات IN , PRE , POST عن موقع العملية المطبقة على الحدين.

تستخدم أغلب المترجمات الصيغة Postfix من أجل حساب قيمة تعبير رياضي وذلك بعد تحويله من الشكل Infix.

القاعدة في تحويل تعبير من Infix إلى Postfix أو إلى Prefix بسيطة إذا عرفنا ترتيب الأسبقية، حيث نعبر عن العمليات الحسابية الجمع والطرح والضرب والقسمة بـ : + ، - . * ، / أما الرفع إلى قوة فنعبر عنها بالإشارة ^ وهكذا فان ترتيب الأسبقية مع الأقواس يكون:

1- ( )

2- ^

3- * ، /

4- - ، +

مثال :

لتحويل التعبير A+B*C إلى الصيغة Postfix:

نعلم أن الضرب له أسبقية على الجمع لذلك نحول قسم التعبير الذي يُ حسب أولاً أي الضرب كما يلي:

- A+(B*C) الأقواس للتوكيد.
- A+(BC*) تحويل الضرب.
- A(BC*)+ تحويل الجمع .

- +*ABC وهو الشكل Postfix.

أما لتحويل نفس التعبير إلى الصيغة Prefix:

نحول قسم التعبير الذي يُ حسب أولاً أي الضرب كما يلي:

- A+(B*C) الأقواس للتوكيد.
- A+(*BC) تحويل الضرب.
- +A(*BC) تحويل الجمع.
- +*ABC وهو الشكل Prefix.

ملاحظات:

1- القاعدة الوحيدة التي يجب أن نتذكرها خلال معالجة التحويل هي أن العمليات بأسبقية أعلى تحول أولاً وبعد ذلك قسم التعبير الذي تم تحويله إلى Postfix أو إلى Prefix يعامل كأنه عامل مفرد.

2- من أجل العمليات من نفس الأسبقية ودون أقواس فانه يتم إنجاز العملية على اليسار أولاً ثم العملية على اليمين ما عدا حالة الرفع إلى قوة حيث تنجز العملية على اليمين أولاً.

3- الفائدة من التحويل :

أ- التخلي عن الأقواس كلياً.

ب- التعرف على ترتيب إنجاز العمليات إذ أن العملية التي تظهر أولاً في السلسلة Postfix هي التي تنفذ أولاً أما العملية التي تظهر أولاً في السلسلة Prefix تنفذ أخيراً إلا أن ترتيب الحدود يبقى نفسه.

2-2-1- خوارزمية التحويل من الصيغة Infix إلى POSTFIX:[24,25]

قبل أن نقدم الخوارزمية لابد لنا من تعريف تابع الأسبقية Takes-precedence الذي يقبل عمليتين كرمزين مفردين ويعيد القيمة TRUE إذا كانت العملية الأولى لها أسبقية على الثانية، ويعيد FALSE في الحالة الأخرى :

BOOL TakesPrecedence (char Operatorstack.top() , char Symbol)

- إذا كان   Operatorstack.top() >= Symbol   فإن   TakesPrecedence =TRUE
- إذا كان   Operatorstack.top() < Symbol   فان   TakesPrecedence =FALSE

<u>أمثلة:</u>

TakesPrecedence (‘*’ , ‘ + ‘)= TRUE

TakesPrecedence (‘+’ , ‘- ‘)= TRUE

TakesPrecedence (‘-’ , ‘ + ‘)= TRUE

TakesPrecedence (‘+’ , ‘ / ‘)= FALSE

TakesPrecedence (‘^’ , ‘ ^ ‘)= FALSE

TakesPrecedence (‘*’ , ‘ / ‘)= TRUE

يمكننا تلخيص الخوارزمية كما يلي:

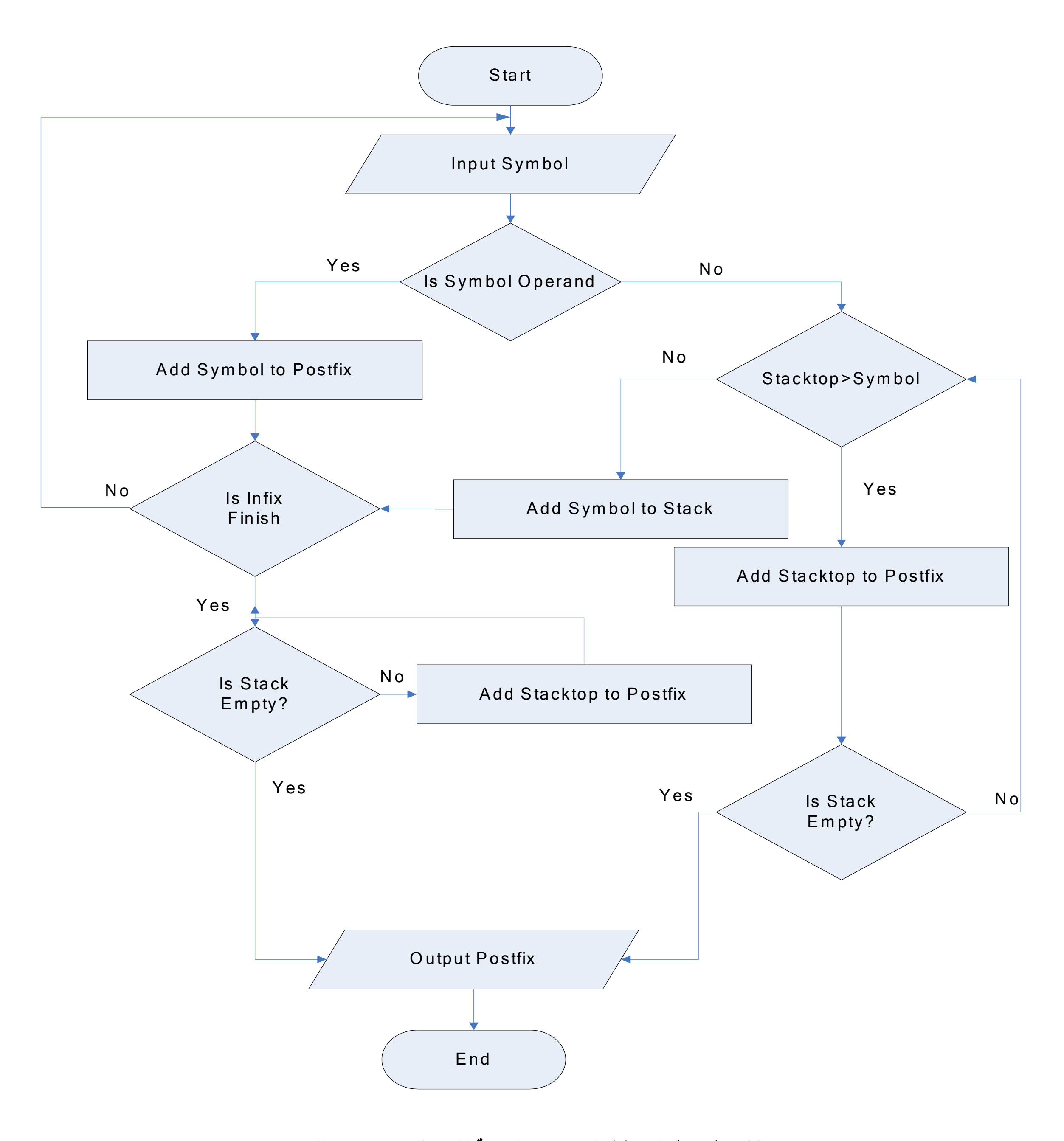
1. طالما لم تنته السلسلة Infix.
2. *اقرأ رمز الدخل Symbol.*
3. إذا كان Symbol حرف ضعه في السلسلة Postfix مباشرة.
4. إذا كان Symbol عملية حسابية افحص المكدس:
   - إذا كان المكدس فارغاً ادفع Symbol إلى المكدس.
   - إذا كان المكدس غير فارغ قارن Symbol مع العملية الموجودة في قمة المكدس:

– إذا كانت العملية Symbol ذات أسبقية أعلى من العملية الموجودة في قمة المكدس، ادفعها إلى المكدس.

– إذا كانـت العمليـة Symbol ذات أسـبقية أدنـى أو مسـاوية للعمليـة الموجـودة فـي قمـة المكـدس، عندئـذ يـتم نـزع العمليـة الموجـودة فـي قمـة المكـدس ووضـعها فـي السلسـلة POSTFIX ، تكرر هذه العمليـة إلـى أن يفرغ المكدس أو نصل إلـى عمليـة ذات أسبقية أدنى من العملية الحالية Symbol، عندئذ يتم دفع العملية Symbol إلى المكدس.

إذا كان Symbol فراغاً هذا يعني أننا وصلنا إلى نهاية سلسلة الدخل، في هذه الحالة يجب فحص المكدس فإذا كان غير فارغ يتم نزع جميع العناصر في المكدس وإدخالها إلى السلسلة Postfix حتى نحصل على مكدس فارغ.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل (2–2):

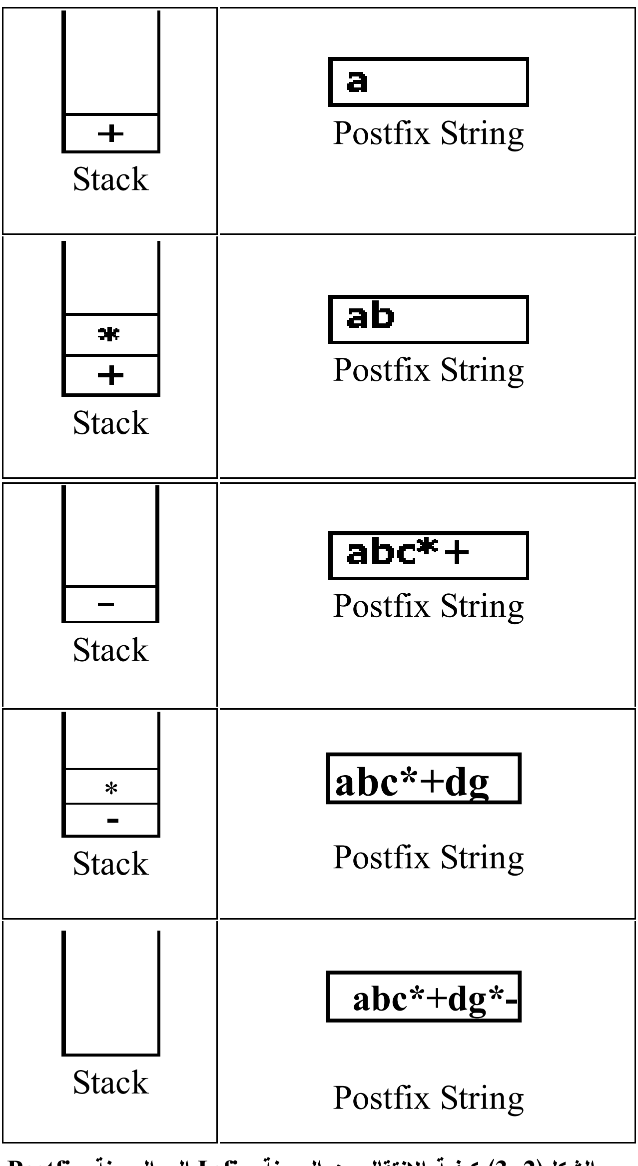


الشكل(2−2) المخطط النهجي لخوارزمية التحويل من Infix إلى Postfix

<u>مثال</u>

Infix String : a+b*c-d*g

إن صيغة Postfix تستنتج كما هوموضح في الشكل(2−3) الذي يبين كيفية التحويل من الصيغة Infix إلى الصيغة Postfix:



الشكل(2-3) كيفية الإنتقال من الصيغة Infix إلى الصيغة Postfix.

يمكن تعديل الخوارزمية السابقة لتلائم وجود الأقواس في التعبير عبر تعريف قواعد الأسبقية الخاصة بالأقواس في التابع Takes-Precedence :

- من اجل أي عملية OP فإن: TakesPrecedence (‘(’ , OP)= FALSE

- من أجل أي عملية ماعدا "(" فإن: TakesPrecedence (OP , ‘(‘)= FALSE

- من أجل أي عملية ماعدا ")" فإن: TakesPrecedence (OP , ‘)’)= TRUE

<u>حالة خاصة</u> :

بما أننا لا نريد نزع القوس المفتوح من المكدس وإضافته إلى السلسلة Postfix ولا نريد أيضاً إدخال القوس المغلق إلى المكدس لذلك نضيف شرط آخر قبل أي عملية دفع للمكدس إذا كان Symbol=')' عندها نقوم بنزع قمة المكدس (وهوالقوس المفتوح) ولا نضيفه إلى السلسلة.

لذلك يكون: TakesPrecedence ('(' , ')')= FALSE

وبالتالي تكون الخوارزمية بعد التعديل كما يلي :

1- طالما لم تنتهي السلسلة Infix.

2- اقرأ رمز الدخل Symbol.

3- إذا كان Symbol حرف ضعه في السلسلة Postfix مباشرة.

4- إذا كان Symbol عملية حسابية افحص المكدس:

أ-إذا كان المكدس فارغ ادفع Symbol إلى المكدس.

ب-إذا كان المكدس غير فارغ قارنٍ Symbol مع العملية الموجودة في قمة المكدس:

- إذا كانت العملية Symbol ذات أسبقية أعلى من العملية الموجودة في قمة المكدس، ادفعها إلى المكدس.

- إذا كانت العملية Symbol ذات أسبقية أدنى أو مساوية للعملية الموجودة في قمة المكدس ، عندئذ يتم نزع العملية الموجودة في قمة المكدس ووضعها في السلسلة Postfix، تكرر هذه العملية إلى أن يفرغ المكدس أو نصل إلى عملية ذات أسبقية أدنى من العملية الحالية Symbol، عندئذ يتم دفع العملية Symbol إلى المكدس.

5- إذا كان Symbol قوساً مفتوحاً ')' ادفعه مباشرة إلى المكدس.

6- إذا كان Symbol قوساً مغلقاً '(' يتم نزع جميع العمليات في قمة المكدس وادخالها إلى السلسلة Postfix وصولا إلى أول قوس مفتوح حيث يتم نزعه ولكن لا يدخل إلى السلسلة Postfix.

7- إذا كان Symbol فراغاً هذا يعني أننا وصلنا إلى نهاية السلسلة Infix ، في هذه الحالة يجب فحص المكدس فإذا لم يكن فارغاً عندها يتم نزع جميع العناصر من المكدس وإدخالها إلى السلسلة Postfix حتى نحصل على المكدس الفارغ.

البرنامج التالي يبين تحويل التعبير المكتوب بالصيغة Infix إلى الصيغة Postfix:

```
// Filename:  convert.cpp
#include <iostream.h>
#include <string.h>
enum bool {false,true}; // Visual C++ يمكن الاستغناء عن هذا التعريف عند استخدام
struct stack
{
     int ptr;
     char arr[50];
     stack()
     {
             ptr=0;
             arr[0];
     }
     char top()
     {
             return arr[ptr];
     }
     void push(char ch)
     {
             ptr++;
             arr[ptr]=ch;
     }
     void pop()
     {
             ptr--;
     }
};
void Convert(char[50], char[50]);
bool IsOperand(char ch);
bool TakesPrecedence(char OperatorA, char OperatorB);
void main()
  {
  char Reply;
  do
     {
             char Infix[50], Postfix[50]="";   // local to this loop
     cout << "Enter an infix expression (e.g. (a+b)/c^2, with no spaces):"<< endl;
     cin >> Infix;
     Convert(Infix, Postfix);
     cout << "The equivalent postfix expression is:" << endl<< Postfix << endl;
     cout << endl << "Do another (y/n)? ";
```

```
    cin >> Reply;
    }
  while (Reply == 'y');
  }
/* Given:  ch   A character.
  Task:   To determine whether ch represents an operand (here understood
        to be a single letter or digit).
  Return: In the function name: true, if ch is an operand, false otherwise.
*/
bool IsOperand(char ch)
  {
  if (((ch >= 'a') && (ch <= 'z')) ||
    ((ch >= 'A') && (ch <= 'Z')) ||
    ((ch >= '0') && (ch <= '9')))
    return true;
  else
    return false;
  }
/* Given:  OperatorA is a character representing an operator or parenthesis.
        OperatorB is a character representing an operator or parenthesis.
  Task: to determine whether OperatorA takes precedence over OperatorB.
  Return: TRUE, if OperatorA takes precedence over OperatorB. */
bool TakesPrecedence(char OperatorA, char OperatorB)
  {
  if (OperatorA == '(')
    return false;
  else if (OperatorB == '(')
    return false;
  else if (OperatorB == ')')
    return true;
  else if ((OperatorA == '^') && (OperatorB == '^'))
    return false;
  else if (OperatorA == '^')
    return true;
  else if (OperatorB == '^')
    return false;
  else if ((OperatorA == '*') || (OperatorA == '/'))
    return true;
  else if ((OperatorB == '*') || (OperatorB == '/'))
    return false;
  else
    return true;
  }
/* Given:  Infix    A string representing an infix expression (no spaces).
  Task:   To find the postfix equivalent of this expression.
  Return: Postfix  A string holding this postfix equivalent.
*/
void Convert(char Infix[50], char Postfix[50])
```

```
{
stack OperatorStack;
char TopSymbol, Symbol;
int L;
for (unsigned k = 0; k < strlen(Infix); k++)
   {
   Symbol = Infix[k];
   if (IsOperand(Symbol))
     {
             L=strlen(Postfix);
             Postfix[L]=Symbol;
             Postfix[L+1]='\0';
     }
   else
      {
while(OperatorStack.ptr)&&(TakesPrecedence(OperatorStack.top(), Symbol)) )
        {
        TopSymbol = OperatorStack.top();
        OperatorStack.pop();
                L=strlen(Postfix);
                Postfix[L]=TopSymbol;
                Postfix[L+1]='\0';
        }
      if (( OperatorStack.ptr) && (Symbol == ')'))
        OperatorStack.pop();   // discard matching (
      else
        OperatorStack.push(Symbol);
      }
   }

   while ( OperatorStack.ptr)
   {
   TopSymbol = OperatorStack.top();
   OperatorStack.pop();
             L=strlen(Postfix);
             Postfix[L]=TopSymbol;
             Postfix[L+1]='\0';
   }
}
```


2-2-2- خوارزمية التحويل من الصيغة Infix إلى Prefix:[24,25]

قبل أن نقدم الخوارزمية لابد لنا من تعديل تابع الأسبقية Takes-precedence :

BOOL TakesPrecedence (char Operatorstack.top() , char Symbol)

- إذا كان Operatorstack.top() > Symbol فإن TakesPrecedence =TRUE
- إذا كان Operatorstack.top() <= Symbol فإن TakesPrecedence =FALSE

يمكننا تلخيص الخوارزمية كما يلي:

1. طالما لم تنتهي السلسلة Infix.
2. اقرأ رمز الدخل Symbol من نهاية السلسلة Infix.
3. إذا كان Symbol حرف ضعه في السلسلة المعكوسة لـ Prefix مباشرة.
4. إذا كان Symbol عملية حسابية افحص المكدس:

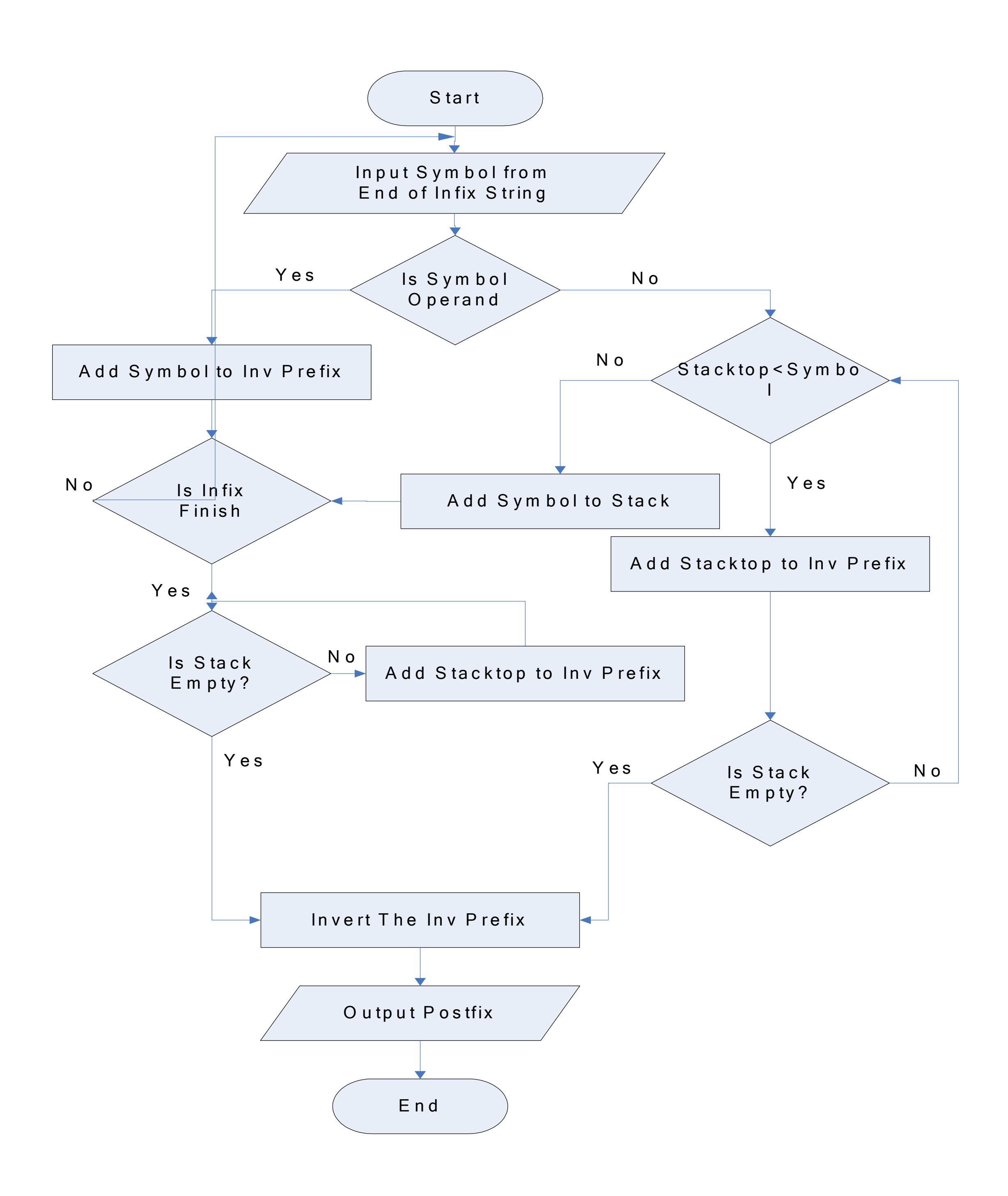
أ- إذا كان المكدس فارغاً ادفع Symbol إلى المكدس.

ب- إذا لم يكن المكدس فارغاً قارن Symbol مع العملية الموجودة في قمة المكدس:

- إذا كانت العملية Symbol ذات أسبقية أعلى أو مساوية للعملية الموجودة في قمة المكدس، ادفعها إلى المكدس.
- إذا كانت العملية Symbol ذات أسبقية أدنى من العملية الموجودة في قمة المكدس، عندئذ يتم نزع العملية الموجودة في قمة المكدس ووضعها في السلسلة المعكوسة لـ Prefix ، تكرر هذه العملية إلى أن يفرغ المكدس أو نصل إلى عملية ذات أسبقية أدنى أو مساوية للعملية الحالية Symbol، عندئذ يتم دفع العملية Symbol إلى المكدس.

5. إذا وصلنا إلى بداية سلسلة الدخل في هذه الحالة يجب فحص المكدس فإذا لم يكن المكدس فارغاً يتم نزع جميع العناصر في المكدس وإدخالها إلى السلسلة المعكوسة لـ Prefix حتى نحصل على المكدس الفارغ.
6. نقوم بعكس السلسلة الناتجة فنحصل على السلسلة Prefix المطلوبة.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل (2-4):

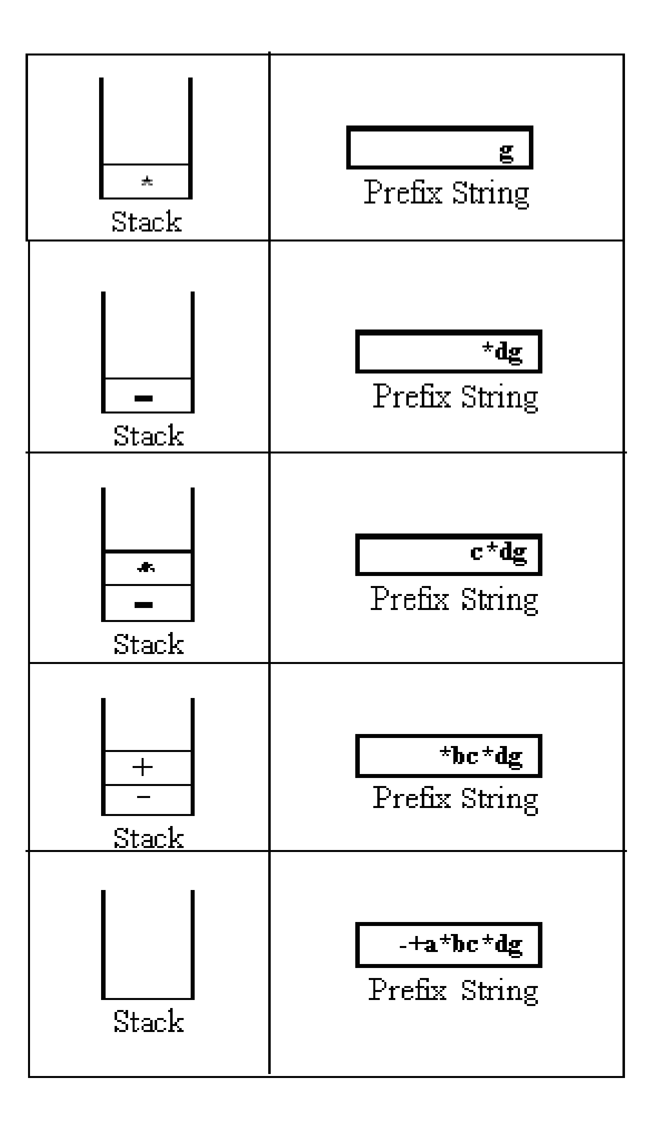


الشكل(2-4) المخطط النهجي لخوارزمية التحويل من Infix إلى Prefix

مثال:

Infix String: a+b*c-d*g

إن صيغة Prefix تستنتج كما هوموضح في الشكل(2-5):



الشكل(2-5) الإنتقال من الصيغة Infix إلى الصيغة Prefix.

يمكن تعديل الخوارزمية السابقة لتلائم وجود الأقواس في التعبير عبر تعريف قواعد الأسبقية الخاصة بالأقواس في التابع Takes-Precedence :

- من أجل أي عملية OP فإن: TakesPrecedence (')' , OP)= FALSE

- من أجل أي عملية ماعدا ")" فإن: TakesPrecedence (OP , ') ')= FALSE

- من أجل أي عملية ماعدا "(" فإن: TakesPrecedence (OP , '(')= TRUE

<u>حالة خاصة :</u>

بما أننا لا نريد نزع القوس المغلق من المكدس وإضافته إلى السلسلة Prefix ولا نريد أيضاً إدخال القوس المفتوح إلى المكدس لذلك نضيف شرط آخر قبل أي عملية دفع للمكدس إذا كان Symbol='(' عندها نقوم بنزع قمة المكدس (وهو القوس المغلق) ولا نضيفه إلى السلسلة.

لذلك يكون: TakesPrecedence (')' , '(')= FALSE

وبالتالي تكون الخوارزمية بعد التعديل كما يلي :

1- طالما لم تنتهي السلسلة Infix.

2- اقرأ رمز الدخل Symbol من نهاية السلسلة Infix.

3- إذا كان Symbol حرف ضعه في السلسلة المعكوسة لـ Prefix مباشرة.

4- إذا كان Symbol عملية حسابية افحص المكدس:

أ-إذا كان المكدس فارغ ادفع Symbol إلى المكدس.

ب-إذا لم يكن المكدس فارغاً قارنٍ Symbol مع العملية الموجودة في قمة المكدس:

- إذا كانت العملية Symbol ذات أسبقية أعلى أو مساوية للعملية الموجودة في قمة المكدس، ادفعها إلى المكدس.

- إذا كانت العملية Symbol ذات أسبقية أدنى من العملية الموجودة في قمة المكدس، عندئذ يتم نزع العملية الموجودة في قمة المكدس ووضعها في السلسلة المعكوسة لـ Prefix ، تكرر هذه العملية إلى أن يفرغ المكدس أو نصل إلى عملية ذات أسبقية أدنى أو مساوية للعملية الحالية Symbol، عندئذ يتم دفع العملية Symbol إلى المكدس.

5- إذا كان Symbol قوساً مغلقاً ')' ادفعه مباشرة إلى المكدس .

6- إذا كان Symbolقوساً مفتوحاً ')' يتم نزع جميع العمليات في قمة المكدس وإدخالها إلى السلسلة المعكوسة لـ Prefixوصولاً إلى أول قوس مغلقحيث يتم نزعه ولكن لا يُ دخل إلى السلسلة المعكوسة لـ Prefix .

7- إذا وصلنا إلى بداية السلسلة Infix في هذه الحالة يجب فحص المكدس فإذا لم يكن المكدس فارغاً عندها يتم نزع جميع العناصر من المكدس وإدخالها إلى السلسلة المعكوسة لـ Prefix حتى نحصل على المكدس الفارغ.

8- نعكس السلسلة الناتجة فنحصل على سلسلة Prefix المطلوبة.

البرنامج التالي يبين تحويل التعبير المكتوب بالصيغة Infix إلى الصيغة Prefix:

```
// Filename:  convert.cpp
#include <iostream.h>
#include <string.h>
enum bool {false,true}; // Visual C++ يمكن الاستغناء عن هذا التعريف عند استخدام
struct stack
{
     int ptr;
     char arr[50];
     stack()
     {
             ptr=0;
             arr[0];
     }
     char top()
     {
             return arr[ptr];
     }
     void push(char ch)
     {
             ptr++;
             arr[ptr]=ch;
     }
     void pop()
     {
             ptr--;
     }
};
void Convert(char[50], char[50]);
bool IsOperand(char ch);
bool TakesPrecedence(char OperatorA, char OperatorB);
```

```
void main()
  {
  char Reply;
  do
    {
          char Infix[50], Prefix[50]="";   // local to this loop
    cout << "Enter an infix expression (e.g. (a+b)/c^2, with no spaces):"<< endl;
    cin >> Infix;
    Convert(Infix, Prefix);
    cout << "The equivalent Prefix expression is:" << endl<< Prefix << endl;
    cout << endl << "Do another (y/n)? ";
    cin >> Reply;
    }
  while (Reply == 'y');
  }
/* Given:  ch   A character.
  Task:   To determine whether ch represents an operand (here understood
       to be a single letter or digit).
  Return: In the function name: true, if ch is an operand, false otherwise.
*/
bool IsOperand(char ch)
  {
  if (((ch >= 'a') && (ch <= 'z')) ||
    ((ch >= 'A') && (ch <= 'Z')) ||
    ((ch >= '0') && (ch <= '9')))
    return true;
  else
    return false;
  }
/* Given:  OperatorA is a character representing an operator or parenthesis.
       OperatorB is a character representing an operator or parenthesis.
  Task: to determine whether OperatorA takes precedence over OperatorB.
  Return: FALSE, if OperatorB takes precedence over or equal OperatorA. */
bool TakesPrecedence(char OperatorA, char OperatorB)
  {
  if (OperatorB== ')')
    return false;
  else if (OperatorA == ')')
    return false;
  else if (OperatorB == '(')
    return true;
  else if ((OperatorA == '^') && (OperatorB == '^'))
    return true;
  else if (OperatorB == '^')
    return true;
  else if (OperatorA == '^')
    return false;
  else if ((OperatorB == '*') || (OperatorB == '/'))
```

```
      return true;
    else if ((OperatorA == '*') || (OperatorA == '/'))
      return false;
    else
      return true;
    }
/* Given:  Infix    A string representing an infix expression (no spaces).
   Task:   To find the prefix equivalent of this expression.
   Return: Prefix  A string holding this prefix equivalent.
*/
void Convert(char Infix[50], char Prefix[50])
    {
    stack OperatorStack;
    char TopSymbol, Symbol;
    int L;
    for (unsigned k = strlen(Infix); k >0; k--)
      {
      Symbol = Infix[k];
      if (IsOperand(Symbol))
        {
                L=strlen(RPrefix);
               RPresfix[L]=Symbol;
                RPrefix[L+1]='\0';
        }
      else
         {
  while(OperatorStack.ptr)&&(TakesPrecedence(OperatorStack.top(), Symbol)) )
           {
           TopSymbol = OperatorStack.top();
           OperatorStack.pop();
                  L=strlen(RPrefix);
                  RPrefix[L]=TopSymbol;
                  RPrefix[L+1]='\0';
           }
         if (( OperatorStack.ptr) && (Symbol == ')'))
           OperatorStack.pop();   // discard matching (
         else
           OperatorStack.push(Symbol);
         }
      }

      while ( OperatorStack.ptr)
      {
      TopSymbol = OperatorStack.top();
      OperatorStack.pop();
                L=strlen(RPrefix);
                RPrefix[L]=TopSymbol;
                RPrefix[L+1]='\0';
```

```
  }
for (unsigned k =0; k < strlen(RPrefix); k++)
  {
   OperatorStack.push(RPrefix[K]);\\ لعكس السلسلة
   }
for (unsigned k =0; k < strlen(RPrefix); k++)
  {
   TopSymbol = OperatorStack.top();
   OperatorStack.pop();
   k=strlen(RPrefix);
   Prefix[k]=TopSymbol;
   Prefix[k+1]='\0';
   }
 }
```

2-3- خوارزمية الانتقال من الصيغة Postfix إلى مخطط تدفق المعطيات DFG:

يمكن تلخيص الخوارزمية كما يلي:[12,16]

1- ننشأ لائحة مترابطة List فارغة أي عدد سجلاتها k=0 .

2- نبدأ من بداية سلسلة الـ Postfix (i=1) .

3- نقرأ رمز الدخل Symbol=Postfix[i].

4- إذا كان رمز الدخل حرف Symbol=Operand عندها نقرأ الرمز التالي i=i+1 .

5- إذا كان رمز الدخل عملية Symbol=Operation ننفذ الخطوات التالية:

أ- ننشأ سجل جديد لهذه العملية k=k+1 .

ب-نضع العملية في حقل Operation لهذا السجل.

ت-نقرأ رمز الدخل Symbol=Postfix[i-1] :

- إذا كان حرف Symbol=Operand نضعه في حقل Var right لهذا السجل ونضع قيمة Null في حقل Next right .
- إذا كان عملية Symbol=Operation نضع في حقل Next right لهذا السجل عنوان السجل السابق Next right=k-1 .

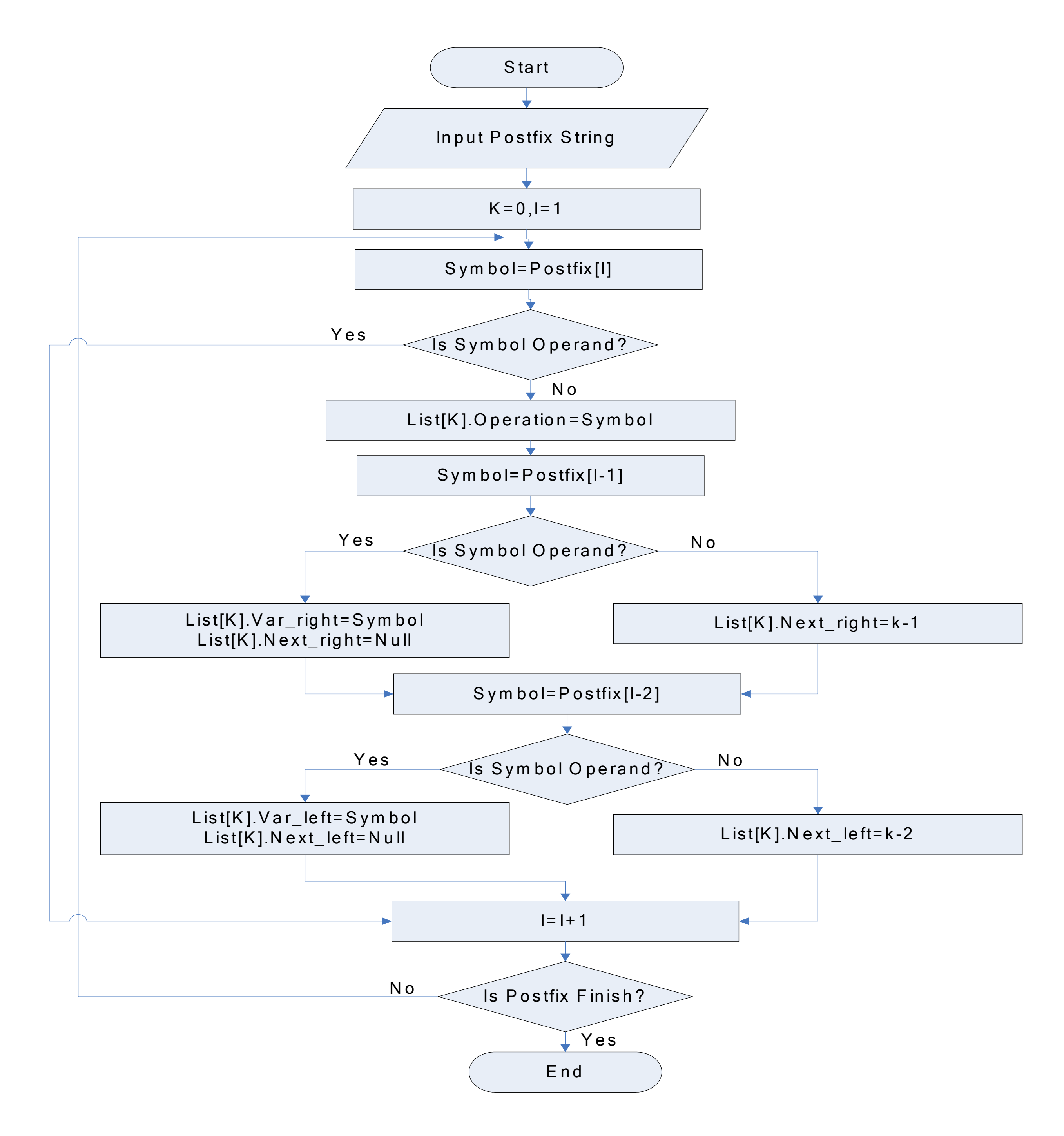
ث-نقرأ رمز الدخل Symbol=Postfix[i-2] :

- إذا كان حرف Symbol=Operand نضعه في حقل Var left لهذا السجل ونضع قيمة Null في حقل Next left.
- إذا كان عملية Symbol=Operation نضع في حقل Next left لهذا السجل عنوان السجل Next left=k-2 .

6- انتقل إلى رمز الدخل التالي في سلسلة الـ Postfix وذلك بجعل i=i+1 ثم كرر الخطوات (2و3و4) حتى نصل إلى نهاية سلسلة الـ Postfix .

7- ضع مؤشر على نهاية هذه اللائحة وذلك بجعل Root=k .

يمكن تلخيص الخوارزمية السابقة في المخطط النهجي الموضح في الشكل(2-6):



الشكل (2-6) المخطط النهجي لخوارزمية التحويل من Postfix إلى DFG

مثال:

من أجل العلاقة التالية Y=a+b*c-d*g فإننا نحصل منها على صيغة Postfix التالية:

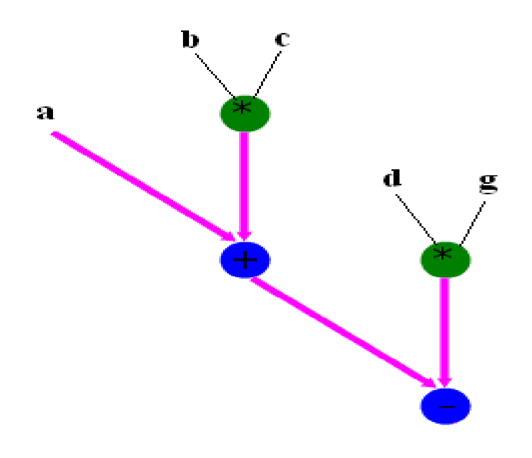
Postfix String: abc*+dg*-

بتطبيق الخوارزمية السابقة على هذه العلاقة نحصل على اللائحة المترابطة التالية الموضحة في الجدول(2-1):

الجدول(2-1) اللائحة المترابطة المعبرة عن DFG.

Rec Num	Var left	Next left	operation	Next right	Var right
1	b	Null	*	Null	c
2	a	Null	+	1	
3	d	Null	*	Null	g
4		2	–	3	

اللائحة المترابطة السابقة تعطينا الشجرة الثنائية التالية الموضحة بالشكل(2-7) وذلك بدون تحديد خطوات زمنية:



الشكل(2-7) الشجرة الثنائية DFG.

الفصل الثالث

مرحلة الجدولة (Scheduling)

3-1- مرحلة الجدولة بدون فرض قيود على الموارد Unconstrained Secheduling:

سنعتمد في هذه المرحلة على خوارزميات الجدولة التالية بدون قيود:

1- Unconstrained ASAP (As Soon As Possible) scheduling algorithm

والتي تعتمد على جدولة العملية في أقرب خطوة زمنية ممكنة بدون وضع قيود على الموارد.

2- Unconstrained ALAP (As Late As Possible) scheduling algorithm

والتي تعتمد على جدولة العملية في أبعد خطوة زمنية ممكنة بدون فرض قيود على الموارد.

3- Unconstrained LIST scheduling algorithm

والتي تعتمد على جدولة العملية اعتماداً على رتل الأولويات بدون فرض قيود على الموارد.

باستخدام خوارزمية الجدولة المناسبة سيكون الدخل عبارة عن لائحة مترابطة تمثل مخطط DFG والخرج عبارة عن لائحة مترابطة أيضاً تمثل ناتج عملية الجدولة لمخطط انسياب المعطيات SDFG وذلك بإضافة الحقل C-Step لتحديد الخطوة الزمنية لتنفيذ العملية الحسابية والحقل Level لتحديد بُعد العملية الحسابية عن معطيات الدخل الأساسية لتصبح بنية السجل كما هو موضح في الشكل (3-1).[13,14,19]]

Var Left	Next Left	Operation	Next Right	Var Right	Level	C-step

الشكل (3-1) بنية السجل في اللائحة المترابطة بعد عملية الجدولة وذلك بدون فرض قيود على معطيات التصميم.

3-1-1- خوارزمية الجدولة Unconstrained ASAP Scheduling بدون قيود:[19]

تعتمد هذه الخوارزمية على جدولة العملية بأقرب خطوة زمنية بغض النظر عن وجود قيود مفروضة على الموارد بمعنى آخر إمكانية وجود وحدات عملياتية كافية في كل خطوة زمنية ، يمكن تلخيص الخوارزمية كما يلي:

1- نبدأ من بداية اللائحة المترابطة التي تعبر عن DFG أي K=1 .

2- إذا كان List[K].Next_right=Null اذهب إلى الخطوة (3) وإلا اذهب إلى الخطوة (5).

3- إذا كان List[K].Next_left=Null عندها اجعل List[K].Level=1 ثم اذهب إلى الخطوة(7).

4- اجعل List[K].Level= List[List[K].Next_left].Level+1 ثم اذهب إلى الخطوة (7).

5- إذا كان List[K].Next_left=Null عندها اجعل
List[K].Level=List[List[K].Next_right].Level+1 ثم اذهب إلى الخطوة (7).

6- اجعل

List[K].Level=Max(List[List[K].Next_right].Level,List[List[K].Next_left].Level)+1

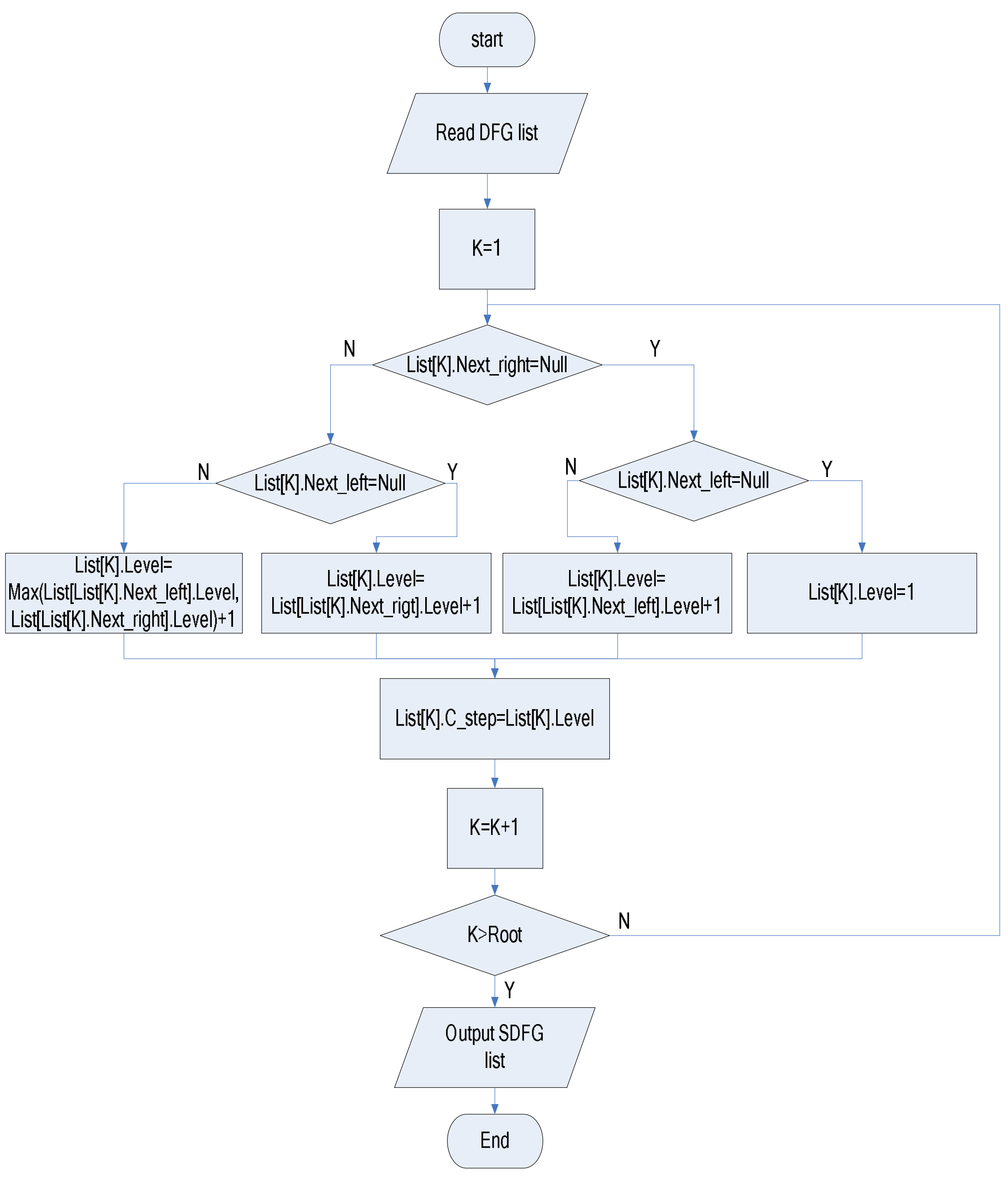
7- List[K].C_step=List[K].Level .

8- K=K+1 .

9- طالما K<=Root أي لم تنتهي اللائحة المترابطة List اذهب إلى الخطوة(2) .

10- اطبع اللائحة المترابطة الجديدة المعبرة عن SDFG باستخدام خوارزمية ASAP بدون قيود.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل(3-2):



الشكل (3-2) المخطط النهجي لخوارزمية **Unconstrained ASAP**.

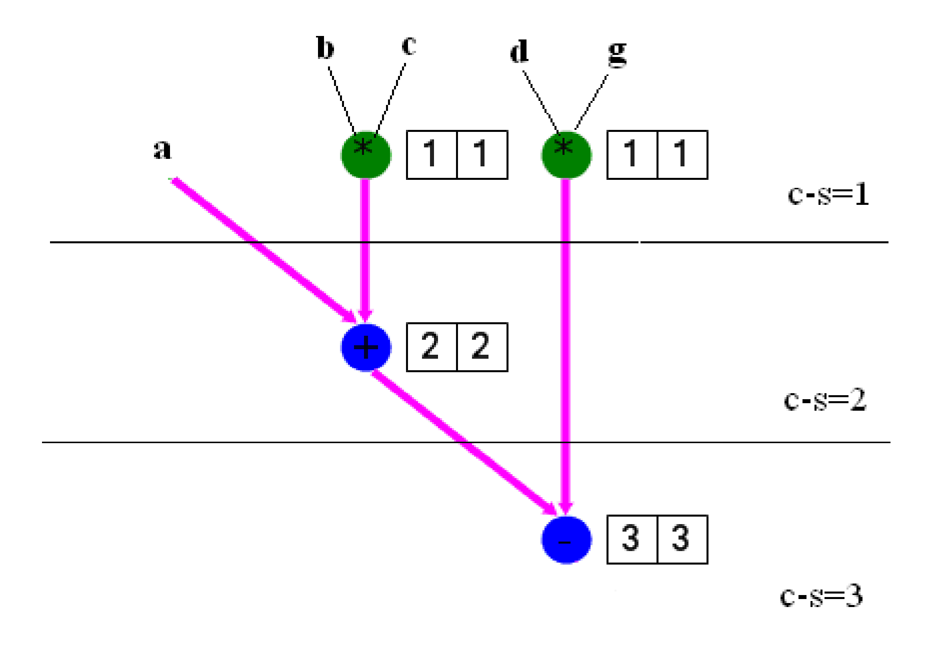
مثال:

من اللائحة المترابطة المعبرة عن مخطط تدفق المعطيات DFG للعلاقة Y=a+b*c-d*g نحصل على اللائحة المترابطة التالية الموضحة في الجدول (3-1) والتي تعبر عن مخطط تدفق المعطيات المجدول SDFG باستخدام ASAP بدون قيود :

الجدول(3-1) اللائحة المترابطة المعبرة عن ASAP بدون قيود

Rec Num	Var left	Next left	operation	Next right	Var right	Level	C-Step
1	b	Null	*	Null	c	1	1
2	a	Null	+	1		2	2
3	d	Null	*	Null	g	1	1
4		2	−	3		3	3

من هذه اللائحة المترابطة نحصل على الشجرة الثنائية التالية الموضحة في الشكل(3-3):



الشكل(3-3) الشجرة الثنائية ASAP بدون قيود

3-1-2- خوارزمية الجدولة Unconstrained ALAP Scheduling بدون قيود:[19]

تعتمد هذه الخوارزمية على جدولة العملية في أبعد خطوة زمنية ممكنة بغض النظر عن وجود قيود مفروضة على النظام، يمكن تلخيص الخوارزمية كما يلي:

1- نبدأ من بداية اللائحة المترابطة التي تعبر عن DFG أي K=1 .

2- إذا كان List[K].Next_right=Null اذهب إلى الخطوة (3) وإلا اذهب إلى الخطوة (5).

3- إذا كان List[K].Next_left=Null عندها اجعل List[K].Level=1 ثم اذهب إلى الخطوة(7).

4- اجعل List[K].Level= List[List[K].Next_left].Level+1 ثم اذهب إلى الخطوة (7).

5- إذا كان List[K].Next_left=Null عندها اجعل

List[K].Level=List[List[K].Next_right].Level+1 ثم اذهب إلى الخطوة (7).

6- اجعل

List[K].Level=Max(List[List[K].Next_right].Level,List[List[K].Next_left].Level)+1

7- K=K+1 .

8- طالما K<=Root أي لم تنتهي اللائحة المترابطة List اذهب إلى الخطوة(2) .

9- K=Root أي ابدأ من نهاية اللائحة المترابطة DFG .

10- List[K].C_step=List[k].Level .

11- إذا كان List[K].Next_right<>Null عندها اجعل

List[List[K].Next_right].C_step=List[K].C_step-1 .

12- إذا كان List[K].Next_left<>Null عندها اجعل

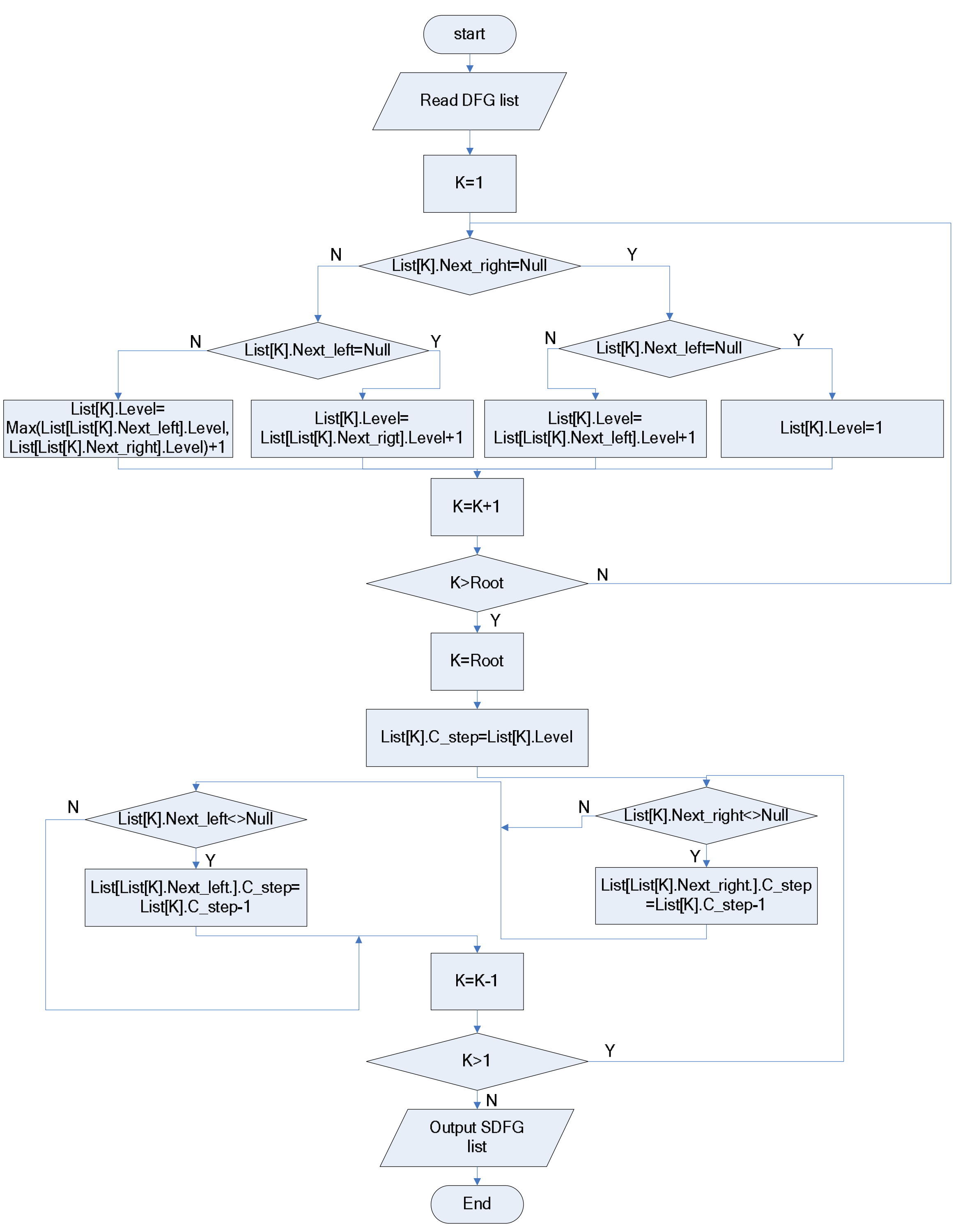
List[List[K].Next_left].C_step=List[K].C_step-1 .

13- K=K-1 .

14- طالما K>=1 أي لم نصل إلى بداية اللائحة كرر الخطوات (11 و12و13).

15- اطبع اللائحة المترابطة الجديدة المعبرة عن SDFG باستخدام خوارزمية ALAP بدون قيود.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل(3-4):



الشكل(3-4) المخطط النهجي لخوارزمية **Unconstrained ALAP.**

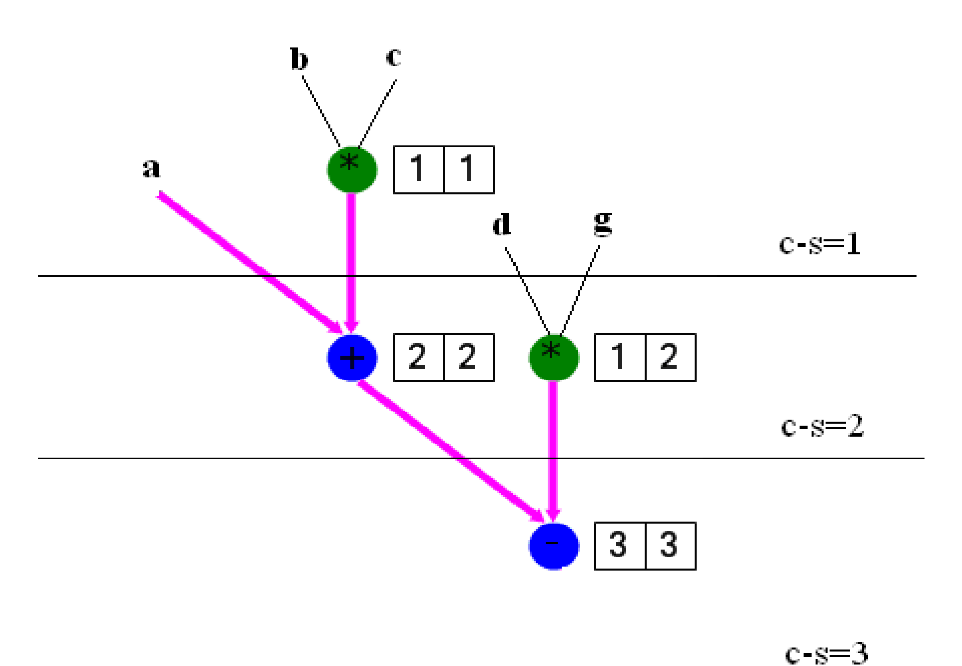
مثال:

من اللائحة المترابطة المعبرة عن مخطط تدفق المعطيات DFG للعلاقة $Y=a+b*c-d*g$ نحصل على اللائحة المترابطة التالية الموضحة في الجدول(3-2) والتي تعبر عن مخطط تدفق المعطيات المجدول SDFG باستخدام ALAP بدون قيود:

الجدول(3-2) اللائحة المترابطة المعبرة عن ALAP بدون قيود

Rec Num	Var left	Next left	operation	Next right	Var right	Level	C-Step
1	b	Null	*	Null	c	1	1
2	a	Null	+	1		2	2
3	d	Null	*	Null	g	1	2
4		2	−	3		3	3

من هذه اللائحة المترابطة نحصل على الشجرة الثنائية التالية الموضحة في الشكل(3-5):



الشكل(3-5) الشجرة الثنائية ALAP بدون قيود

3-1-3- خوارزمية الجدولة Unconstrained LIST Scheduling بدون قيود:[19]

تعتمد هذه الخوارزمية على جدولة العملية اعتماداً على رتل الأولويات بغض النظر عن وجود قيود مفروضة على النظام، يمكن تلخيص الخوارزمية كما يلي:

1- نبدأ من بداية اللائحة المترابطة التي تعبر عن DFG أي K=1 .

2- إذا كان List[K].Next_right=Null اذهب إلى الخطوة (3) وإلا اذهب إلى الخطوة (5).

3- إذا كان List[K].Next_left=Null عندها اجعل List[K].Level=1 ثم اذهب إلى الخطوة(7).

4- اجعل List[K].Level= List[List[K].Next_left].Level+1 ثم اذهب إلى الخطوة (7).

5- إذا كان List[K].Next_left=Null عندها اجعل List[K].Level=List[List[K].Next_right].Level+1 ثم اذهب إلى الخطوة (7).

6- اجعل

List[K].Level=Max(List[List[K].Next_right].Level,List[List[K].Next_left].Level)+1

7- K=K+1 .

8- طالما K<=Root أي لم تنتهي اللائحة المترابطة List اذهب إلى الخطوة(2) .

9- أنشأ رتل أولويات فارغ واجعل i=1 .

10- املأ رتل الأولويات بكل السجلات ذات نفس المستوى أي حقل Level فيها يسأو ي إلى i.

11- اجعل حقل C_step لجميع السجلات في رتل الأولويات مساوياً إلى الحقل Level .

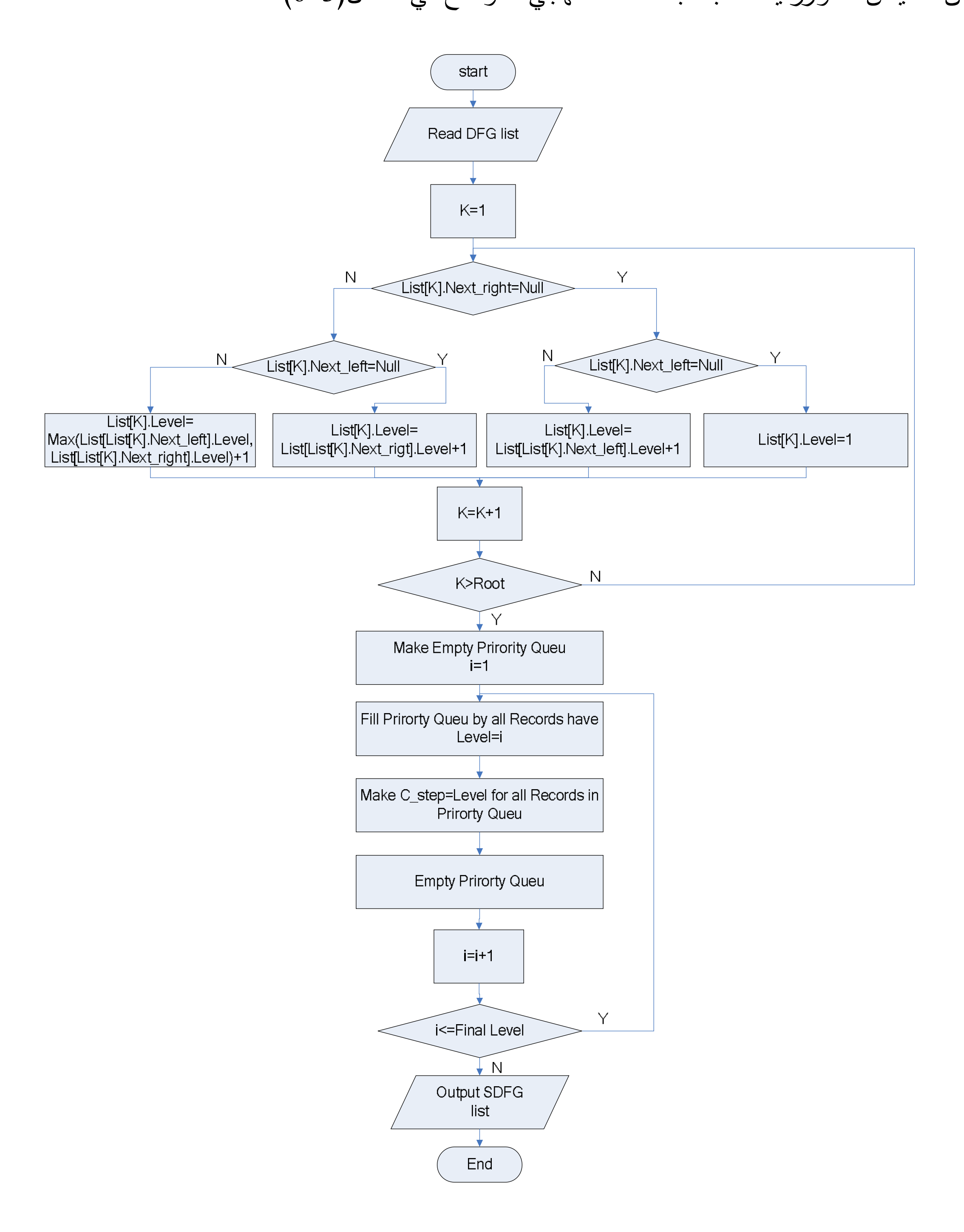
12- أفرغ رتل الأولويات.

13- i=i+1 .

14- طالما توجد سجلات ذات مستويات أعلى أي i<=Final Level اذهب إلى الخطوة(10).

15- اطبع اللائحة المترابطة الجديدة المعبرة عن SDFG باستخدام خوارزمية List بدون قيود.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل(3-6):



الشكل(3-6) المخطط النهجي لخوارزمية Unconstrained List.

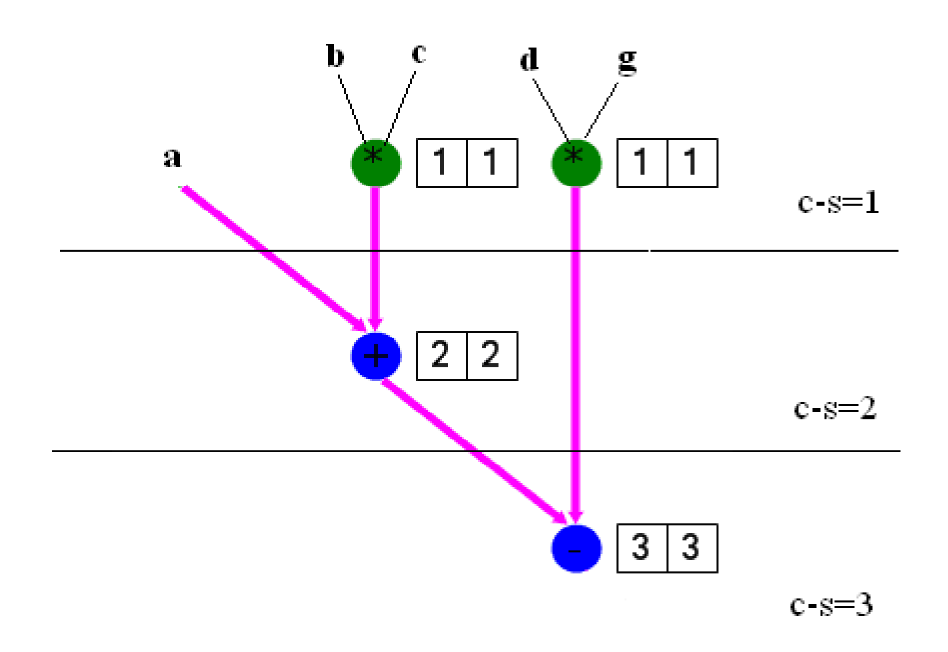
<u>مثال:</u>

من اللائحة المترابطة المعبرة عن مخطط تدفق المعطيات DFG للعلاقة Y=a+b*c-d*g نحصل على اللائحة المترابطة التالية الموضحة في الجدول(3-3) والتي تعبر عن مخطط تدفق المعطيات المجدول SDFG باستخدام List بدون قيود :

الجدول(3-3) اللائحة المترابطة المعبرة عن List بدون قيود

Rec Num	Var left	Next left	operation	Next right	Var right	Level	C-Step
1	b	Null	*	Null	c	1	1
2	a	Null	+	1		2	2
3	d	Null	*	Null	g	1	1
4		2	−	3		3	3

من هذه اللائحة المترابطة نحصل على الشجرة الثنائية التالية الموضحة في الشكل(3-7):



الشكل(3-7) الشجرة الثنائية List بدون قيود

3-2-مرحلة الجدولة مع قيود على الموارد Resource Constrained Secheduling:

سنعتمد في هذه المرحلة على خوارزميات الجدولة التالية مع قيود:

1- Resource Consrained ASAP (As Soon As Possible) scheduling algorithm

والتي تعتمد على جدولة العملية في أقرب خطوة زمنية ممكنة مع وضع قيود على الموارد.

2- Resource Constrained ALAP (As Late As Possible) scheduling algorithm

والتي تعتمد على جدولة العملية في أبعد خطوة زمنية ممكنة مع فرض قيود على الموارد.

3- Resource Constrained LIST scheduling algorithm

والتي تعتمد على جدولة العملية اعتماداً على رتل الأولويات مع فرض قيود على الموارد.

عند وجود قيود سيتم إضافة الحقل Free_Degree إلى بنية السجل الموضح في الشكل (3-1) ليصبح بالشكل (3-8) التالي:[19]

Var Left	Next Left	Operation	Next Right	Var Right	Level	C-step	Free-D

الشكل(3-8) بنية السجل في اللائحة المترابطة بعد عملية الجدولة مع قيود

المقصود بدرجة الحرية Free_Degree هي قابلية حركة العملية بين الخطوات الزمنية المختلفة، عندما تكون Free_D=0 فهذا يعني أن العملية غير قابلة للحركة وإذا فرضت علينا القيود تحريكها فهذا يعني زيادة عدد الخطوات الزمنية للأداة المنطقية .

3-2-1- خوارزمية الجدولة Resource Constrained ASAP Scheduling:

المقصود بالقيود هو وضع شروط على معطيات التصميم مثل تحديد عدد الوحدات العملياتية التي يسمح باستخدامها في كل خطوة زمنية على سبيل المثال وجود ضارب واحد وجامعين في الأداة المنطقية سيفرض علينا جدولة العملية حسب ASAP آخذين بعين الاعتبار عدم السماح لأكثر من عملية ضرب واحدة وأكثر من عمليتي جمع في الخطوة الزمنية الواحدة.[12,17]

ومنه يمكن تلخيص الخوارزمية كما يلي:

1- جدول حسب خوارزميتي ASAP و ALAP بدون قيود من أجل حساب حقل Free_D لكل السجلات في اللائحة المترابطة وذلك اعتماداً على العلاقة التالية:

List[K].Free_D=List[k].C_step(ALAP)-List[K].C_step(ASAP)

2- جدول حسب خوارزمية ASAP بدون قيود .

3- نبدأ من بداية اللائحة المترابطة List وذلك بجعل K=1 .

4- إذا لم يكن هناك قيود مفروضة على العملية للسجل K عندها اذهب إلى الخطوة(9).

5- إذا كان هناك موارد كافية للعملية عندها اذهب إلى الخطوة (9).

6- إذا كان List[K].Free_D<>0 عندها اذهب إلى الخطوة (8).

7- عدل الخطوات الزمنية لكل السجلات التالية في اللائحة المترابطة (C_step=C_step+1) بإضافة خطوة واحدة لها ثم اذهب إلى الخطوة(9).

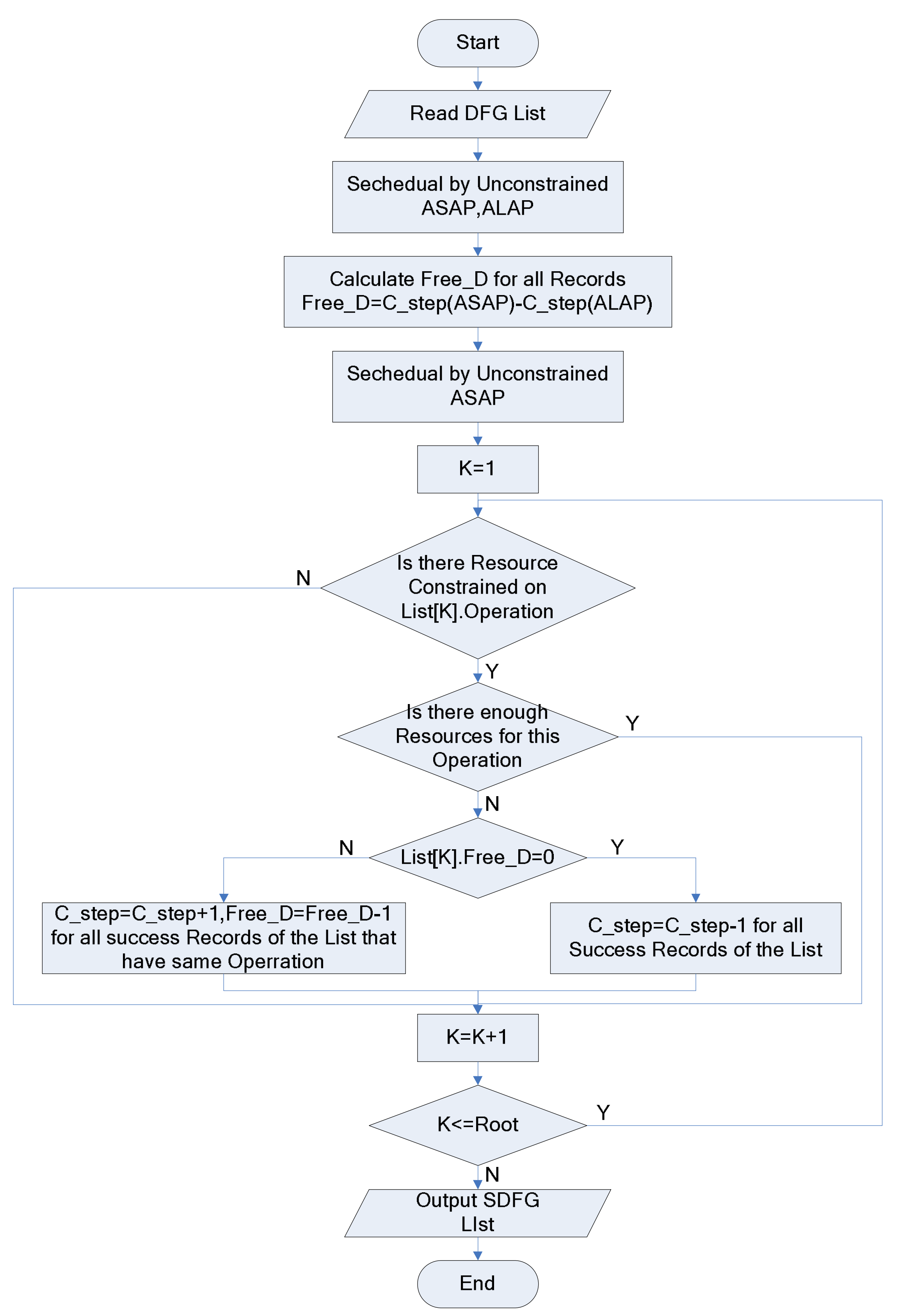
8- عدل الخطوات الزمنية ودرجات الحرية لكل السجلات التالية ذات العمليات المماثلة للعملية المقيدة بإضافة خطوة زمنية واحدة (C_step=C_step+1) وإنقاص درجة حرية واحدة (Free_D=Free_D-1) ثم اذهب إلى الخطوة (9).

9- K=K+1 .

10- طالما لم تنتهي اللائحة المترابطة أي K<=Root اذهب إلى الخطوة (4).

11 - اطبع اللائحة المترابطة الجديدة المعبرة عن SDFG باستخدام خوارزمية ASAP مع قيود.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل(3-9):



الشكل(3-9) المخطط النهجي لخوارزمية **Resource Constrained ASAP**.

مثال:

بعد تطبيق خوارزميتا ASAP و ALAP بدون قيود على العلاقة Y=a+b*c-d*g نستطيع من أجل اللائحة احتساب حقل Free_D للائحة المترابطة كما يلي:

Free_D=[C_S(ALAP)]-[C_S(ASAP)]

وبالتالي تصبح اللائحة المترابطة التي تعبر عن مخطط تدفق المعطيات المجدول SDFG باستخدام خوارزمية ALAP المهيأة لخوارزميات الجدولة بقيود موضحة في الجدول(3–4):

الجدول(3–4) اللائحة المترابطة المعبرة عن ALAP المهيأة لخوارزميات الجدولة بقيود

Rec Num	Var left	Next left	operation	Next right	Var right	Level	C-Step	Free-D
1	b	Null	*	Null	c	1	1	0
2	a	Null	+	1		2	2	0
3	d	Null	*	Null	g	1	2	1
4		2	–	3		3	3	0

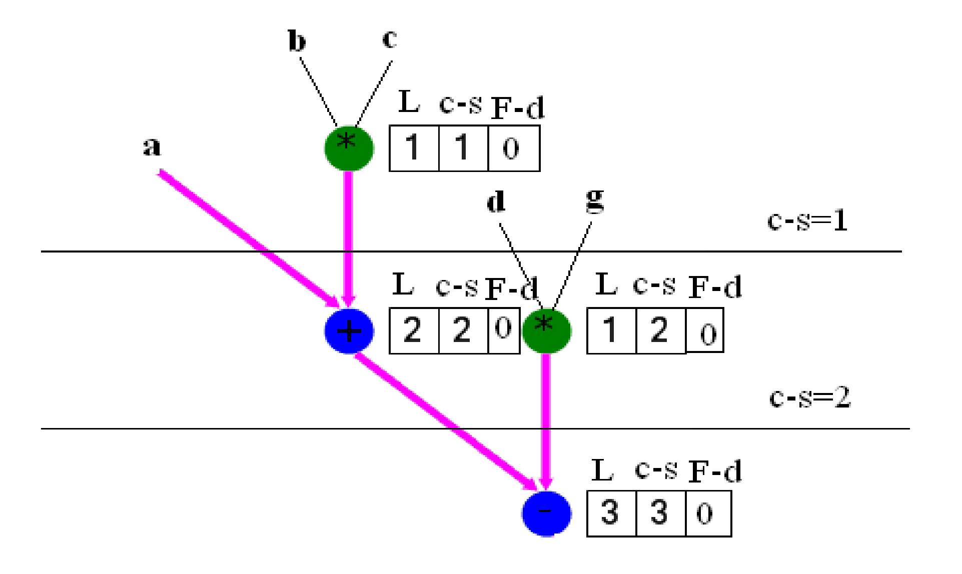
نلاحظ أن العملية الثالثة فقط قابلة للحركة بمقدار خطوة زمنية واحدة فقط أما بقية العمليات غير قابلة للحركة.

نستطيع الآن تطبيق خوارزمية ASAP مع قيود ، سنفترض أنه يوجد قيد على عدد الضوارب بحيث لا يسمح لنا بتنفيذ أكثر من عملية ضرب واحدة في كل خطوة زمنية ، عندها ستكون اللائحة المترابطة المعبرة عن مخطط تدفق المعطيات المجدول SDFG باستخدام ASAP مع قيود موضحة في الجدول(3–5):

الجدول(3–5) اللائحة المترابطة المعبرة عن ASAP بقيود

Rec Num	Var left	Next left	operation	Next right	Var right	Level	C-Step	Free-D
1	b	Null	*	Null	c	1	1	0
2	a	Null	+	1		2	2	0
3	d	Null	*	Null	g	1	2	0
4		2	–	3		3	3	0

ومنه الشجرة الثنائية المعبرة عن هذه اللائحة المترابطة موضحة في الشكل(3–10):



الشكل(3-10) الشجرة الثنائية ASAP بقيود

3-2-2- خوارزمية الجدولة Resource Constrained ALAP Scheduling:

عند جدولة العملية حسب ALAP سنأخذ بعين الاعتبار القيود المفروضة على النظام من حيث عدد الوحدات العملياتية وبالتالي يمكن تلخيص الخوارزمية كما يلي:[12,17]

1- جدول حسب خوارزميتي ASAP و ALAP بدون قيود من أجل حساب حقل Free_D لكل السجلات في اللائحة المترابطة وذلك اعتماداً على العلاقة التالية:

List[K].Free_D=List[k].C_step(ALAP)-List[K].C_step(ASAP)

2- جدول حسب خوارزمية ALAP بدون قيود .

3- نبدأ من بداية اللائحة المترابطة List وذلك بجعل K=1 .

4- إذا لم يكن هناك قيود مفروضة على العملية للسجل K عندها اذهب إلى الخطوة(7).

5- إذا كان هناك موارد كافية للعملية عندها اذهب إلى الخطوة (7).

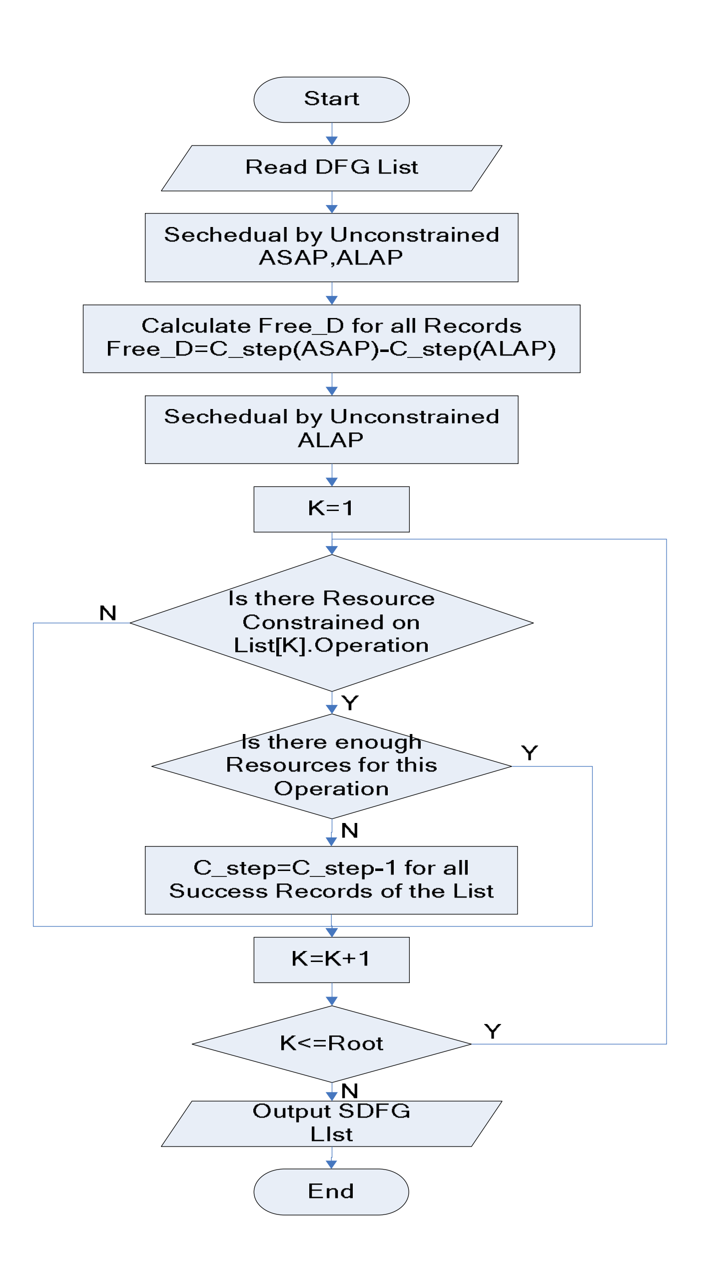
6- عدل الخطوات الزمنية لكل السجلات التالية في اللائحة المترابطة (C_step=C_step+1) بإضافة خطوة واحدة لها ثم اذهب إلى الخطوة(7).

7- K=K+1 .

8- طالما لم تنتهي اللائحة المترابطة أي K<=Root اذهب إلى الخطوة (4).

9 - اطبع اللائحة المترابطة الجديدة المعبرة عن SDFG باستخدام خوارزمية ALAP مع قيود.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل(3-11):



الشكل(3-11) المخطط النهجي لخوارزمية Resource Constrained ALAP.

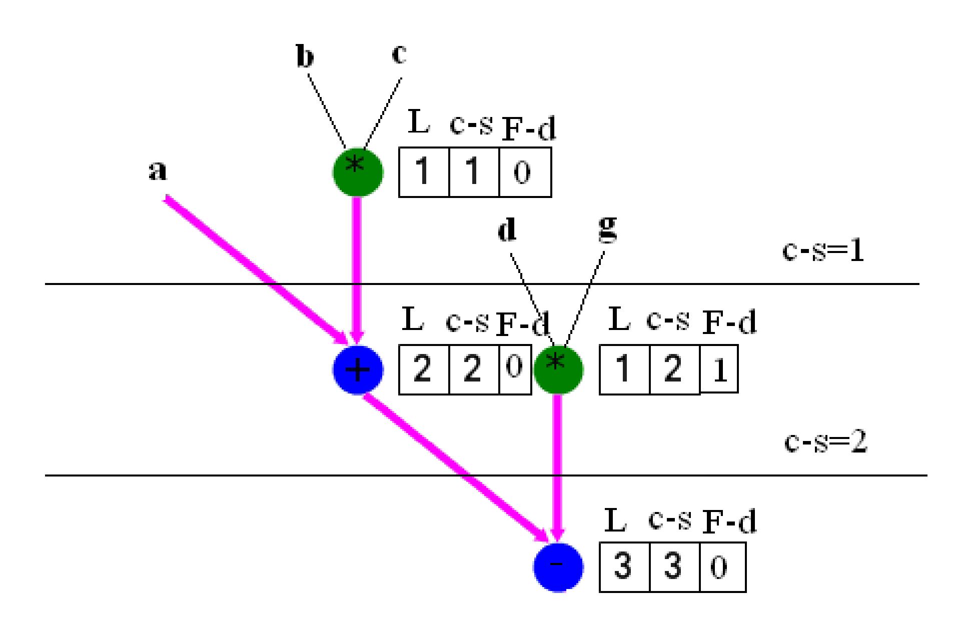
<u>مثال:</u>

بعد تطبيق خوارزميتا ASAP و ALAP بدون قيود على العلاقة Y=a+b*c-d*g وحساب حقل Free_D للعمليات الحسابية نستطيع أن نطبق خوارزمية ALAP مع قيود ، سنفترض أنه يوجد قيد على عدد الضوارب بحيث لا يسمح لنا بتنفيذ أكثر من عملية ضرب واحدة في كل خطوة زمنية ، عندها ستكون اللائحة المترابطة المعبرة عن مخطط تدفق المعطيات المجدول SDFG باستخدام ALAP مع قيود موضحة في الجدول(7):

الجدول(3-6) اللائحة المترابطة المعبرة عن ALAP بقيود

Rec Num	Var left	Next left	operation	Next right	Var right	Level	C-Step	Free-D
1	b	Null	*	Null	c	1	1	0
2	a	Null	+	1		2	2	0
3	d	Null	*	Null	g	1	2	1
4		2	–	3		3	3	0

ومنه الشجرة الثنائية المعبرة عن هذه اللائحة المترابطة موضحة في الشكل (3-12):



الشكل(3-12) الشجرة الثنائية ALAP بقيود

3-2-3- خوارزمية الجدولة Resource Constrained LIST Scheduling:

عند جدولة العملية حسب List سنأخذ بعين الاعتبار القيود المفروضة على النظام من حيث عدد الوحدات العملياتية وبالتالي يمكن تلخيص الخوارزمية كما يلي:[12,17]

1- اقرأ اللائحة المترابطة التي تعبر عن DFG .

2- احسب حقل Level لكل السجلات في اللائحة المترابطة.

3- أنشأ رتل أولويات فارغ .

4- املأ رتل الأولويات بكل السجلات التي فيها الحقل Level=1 .

5- ابدأ من بداية رتل الأولويات أي k=1 .

6- إذا لم يكن هناك قيود مفروضة على العملية للسجل K عندها اذهب إلى الخطوة(9).

7- إذا كان هناك موارد كافية للعملية عندها اذهب إلى الخطوة (9).

8- احسب الحقل C_step للسجل k وفق العلاقة: C_step(k)=C_step(k-1)+1 ثم اذهب إلى الخطوة(10).

9- احسب الحقل C_step للسجل k وفق العلاقة: C_step(k)=Level(k) .

10- هل انتهى رتل الأولويات.

11- i=2 .

12- املأ رتل الأولويات بكل السجلات التي فيها الحقل Level=i .

13- احسب حقل C_step لكل السجلات في رتل الأولويات وفق العلاقة:

C_step(k)=Max{C_step(N_Right(k)),C_step(N_Left(k))}+1

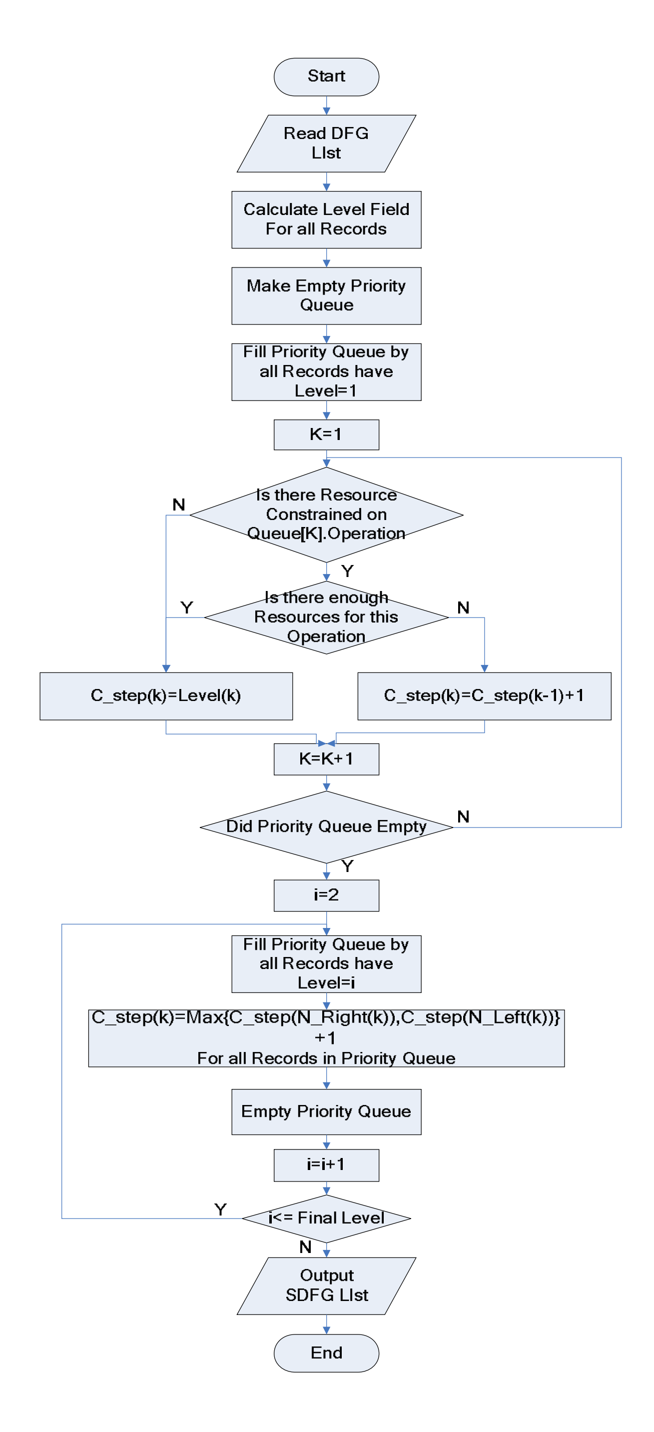
14- افرغ رتل الأولويات.

15- i=i+1 .

14- طالما توجد سجلات ذات مستويات أعلى أي i<=Final Level اذهب إلى الخطوة(12).

15- اطبع اللائحة المترابطة الجديدة المعبرة عن SDFG باستخدام خوارزمية List مع قيود.

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل(3-13):



الشكل(3-13) المخطط النهجي لخوارزمية Resource Constrained List.

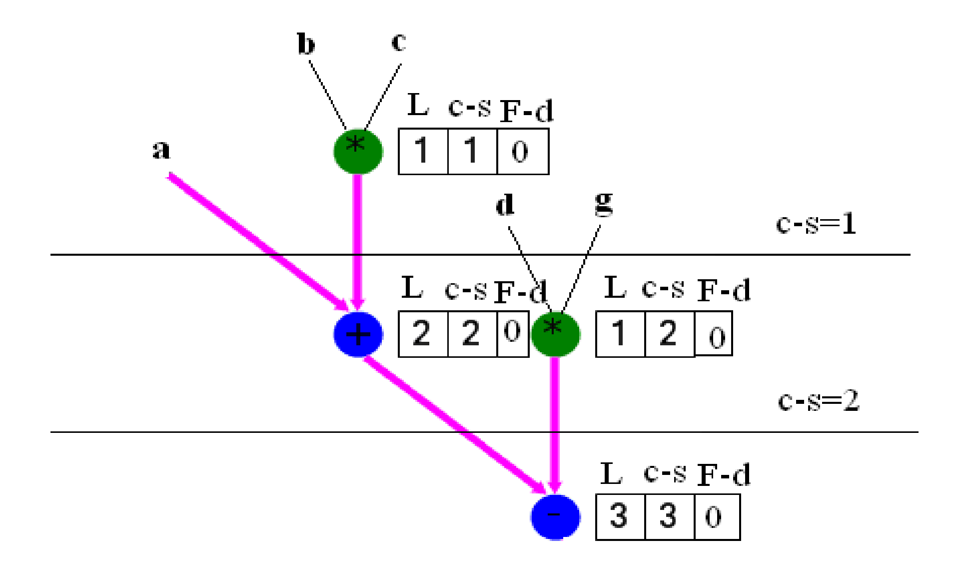
مثال:

سنفترض أنه يوجد قيد على عدد الضوارب بحيث لا يسمح لنا بتنفيذ أكثر من عملية ضرب واحدة في كل خطوة زمنية، عندها ستكون اللائحة المترابطة المعبرة عن مخطط تدفق المعطيات المجدول SDFG باستخدام List مع قيود موضحة في الجدول(3-7):

الجدول(3-7) اللائحة المترابطة المعبرة عن List بقيود

Rec Num	Var left	Next left	operation	Next right	Var right	Level	C-Step	Free-D
1	b	Null	*	Null	c	1	1	0
2	a	Null	+	1		2	2	0
3	d	Null	*	Null	g	1	2	0
4		2	−	3		3	3	0

ومنه الشجرة الثنائية المعبرة عن هذه اللائحة المترابطة موضحة في الشكل(3-14):



الشكل(3-14) الشجرة الثنائية List بقيود

سيتم التركيز على رسم DFG قبل وبعد الجدولة مع حساب درجة تعقيد خوارزميات الترجمة والجدولة وخوارزميات الرسم وذلك عبر حساب أزمنة التنفيذ.

تم الاستفادة من النتائج التي توصلنا إليها في اختيار اللغة البرمجية الأنسب لمتابعة العمل في بناء الأداة المنطقية كما هو مبين لاحقاً.

الفصل الرابع

دراسة مقارنة وتقويم للخوارزميات المستخدمة في مرحلتي

الترجمة والجدولة من مراحل بناء الأداة الرقمية

4-1- مقدمة عن لغات البرمجة المستخدمة:

تمت الدراسة من أجل العلاقة العامة التالية المشتقة من الصيغة العامة للمرشح الرقمي IIR :

$$y(n) = \sum_{k=1}^{N} a_k * b_k$$

حيث سنناقش مرحلتي الترجمة والجدولة وذلك بثلاث لغات برمجية وهي: #Delphi –C– Java وذلك من أجل تقويم أداء خوارزميات الترجمة والجدولة في اللغات الثلاثة المختارة تمهيداً لاختيار اللغة الأنسب من ناحية الأداء بغية المتابعة فيها في المراحل التالية من مراحل بناء النظام. في هذه الحالة فإنه سوف يتم التركيز على الناحية البرمجية في بناء الأداة المطلوبة ولن نتطرق، إلى التحقق من مطابقة التوصيف السلوكي VHDL للتوصيف البنيوي في مستوى RTL [2].

هذا الفصل يتعرض لمقارنة اللغات البرمجية الثلاثة المختارة من خلال منحنيات بيانية تبين زمن المعالجة وكمية المعطيات التي تتم معالجتها كذلك تم توضيح الأسباب التي تجعل من اللغة المختارة الأنسب سواء من حيث سرعة التنفيذ والقيمة العظمى التي تستطيع الوصول إليها وجودة الشجرة الثنائية الممثلة للنظام (كما ورد في مجلة بحوث جامعة حلب سلسلة العلوم الهندسية العدد /66/ لعام 2009).

إن اختيار لغات البرمجة المذكورة يهدف إلى تغطية ثلاثة أصناف من لغات البرمجة وهي كما يلي:

- لغة #C والتي هي مشتقة من لغة C وتمثل لغات البرمجة متوسطة المستوى فهي متوسطة بين لغة المعالج (Assembly) واللغات عالية المستوى.
- لغة Delphi والتي هي مشتقة من لغة Pascal وتعتبر من لغات البرمجة عالية المستوى التقليدية.
- لغة Java وهي من اللغات عالية المستوى المتقدمة والتي تسمح بإنشاء تطبيقات Internet.

تم تطبيق البرامج الثلاثة على نفس الحاسب يتمتع بالمواصفات التالية:

. Intel® Celeron® D CPU 3.46GHz,120 GByte HDD,1.0 GByte RAM

في بداية البرنامج ستحدد قيمة N للعلاقة السابقة وتنفذ الخطوات التالية من أجل i=1….N:

1. تحويل معادلة الدخل من الشكل Infix إلى الشكل Postfix لتحديد أو لويات العمليات وحذف الأقواس إن وجدت .

2. تحويل الصيغة Postfix لمعادلة الدخل إلى مخطط انسياب المعطيات DFG على شكل شجرة ثنائية Binary Tree مؤلفة من لائحة مترابطة أو مصفوفة.

3. رسم الشجرة الثنائية التي تعبر عن اللائحة المترابطة DFG الناتجة قبل الجدولة.

4. تحويل مخطط انسياب المعطيات DFG إلى مخطط انسياب معطيات مجدول SDFG باستخدام خوارزمية ASAP بدون قيود .

5. رسم الشجرة الثنائية التي تعبر عن SDFG باستخدام ASAP بدون قيود.

6. تحويل مخطط انسياب المعطيات DFG إلى مخطط انسياب معطيات مجدول SDFG باستخدام خوارزمية ALAP بدون قيود .

7. رسم الشجرة الثنائية التي تعبر عن SDFG باستخدام ALAP بدون قيود.

8. تحويل مخطط انسياب المعطيات DFG إلى مخطط انسياب معطيات مجدول SDFG باستخدام خوارزمية LIST بدون قيود .

9. رسم الشجرة الثنائية التي تعبر عن SDFG باستخدام List بدون قيود.

10. تحويل مخطط انسياب المعطيات DFG إلى مخطط انسياب معطيات مجدول SDFG باستخدام خوارزمية ASAP مع قيود .

11. رسم الشجرة الثنائية التي تعبر عن SDFG باستخدام ASAP مع قيود.

12. تحويل مخطط انسياب المعطيات DFG إلى مخطط انسياب معطيات مجدول SDFG باستخدام خوارزمية ALAP مع قيود .

13. رسم الشجرة الثنائية التي تعبر عن SDFG باستخدام ALAP مع قيود.

14. تحويل مخطط انسياب المعطيات DFG إلى مخطط انسياب معطيات مجدول SDFG باستخدام خوارزمية List مع قيود .

15. رسم الشجرة الثنائية التي تعبر عن SDFG باستخدام List مع قيود.

16. حساب الأزمنة التالية:

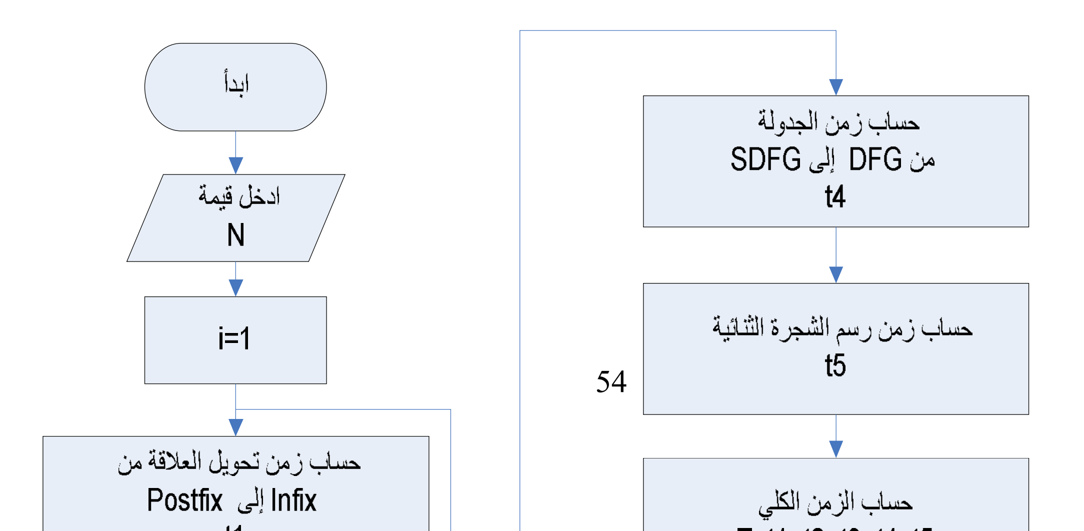
- زمن التحويل من الصيغة Infix إلى الصيغة Postfix .
- زمن التحويل من الصيغة Postfix إلى مخطط تدفق المعطيات DFG .
- زمن رسم الشجرة الثنائية باستخدام مخطط تدفق المعطيات DFG .
- زمن التحويل من مخطط تدفق المعطيات DFG إلى مخطط تدفق مخطط تدفق المعطيات

المجدول SDFG باستخدام إحدى خوارزميات الجدولة.

- زمن رسم الشجرة الثنائية المعبرة عن مخطط تدفق المعطيات المجدول SDFG.
- الزمن الكلي لهذه العمليات .

17. رسم المنحني الزمني الخاص بهذه الأزمنة من أجل i=1....N لحساب درجة تعقيد خوارزميات الترجمة والجدولة بالنسبة للغات البرمجة المختارة ليتم اختيار اللغة الأنسب لمتابعة العمل في خوارزميات النظم الموزعة .

الخطوات السابقة موضحة في المخطط النهجي الموضح في الشكل(4-1):



الشكل(4-1) المخطط النهجي لمرحلتي الترجمة والجدولة

4-2- مثال تطبيقي عن مرحلتي الترجمة والجدولة:

لتوضيح خطوات العمل السابقة سنأخذ المثال التالي:

1- من أجل N=10 تصبح العلاقة حسب صيغة Infix بالشكل التالي:

$$Y=a_1*b_1+a_2*b_2+\ldots\ldots+a_{10}*b_{10}$$

2- حسب خوارزمية التحويل من Infix إلى Postfix تصبح العلاقة كما هو موضح في الشكل(4-2):

حالة المكدس	علاقة الـ Postfix
*	a_1

Stack	Output
+	a_1b_1*
* +	a_1b_1*a_2
+	a_1b_1*a_2b_2*+
* +	a_1b_1*a_2b_2*+a_3
+	a_1b_1*a_2b_2*+a_3b_3*+
* +	a_1b_1*a_2b_2*+a_3b_3*+a_4
+	a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+
* +	a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5

Stack	Output
+	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+$
* +	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6$
+	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+$
* +	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7$
+	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+$
* +	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+$ a_8
+	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+$ a_8b_8*+
* +	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+$ $a_8b_8*+a_9$

Stack	Postfix
+	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+ a_8b_8*+a_9b_9*+$
* +	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+ a_8b_8*+a_9b_9*+a_{10}$
	$a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+ a_8b_8*+a_9b_9*+a_{10}b_{10}*+$

الشكل(4-2) الانتقال من الصيغ Infix إلى الصيغة Postfix

وبالتالي حصلنا على صيغة الـ Postfix لمعادلة الدخل التالية:

$$Y= a_1b_1*a_2b_2*+a_3b_3*+a_4b_4*+a_5b_5*+a_6b_6*+a_7b_7*+ a_8b_8*+a_9b_9*+a_{10}b_{10}*+$$

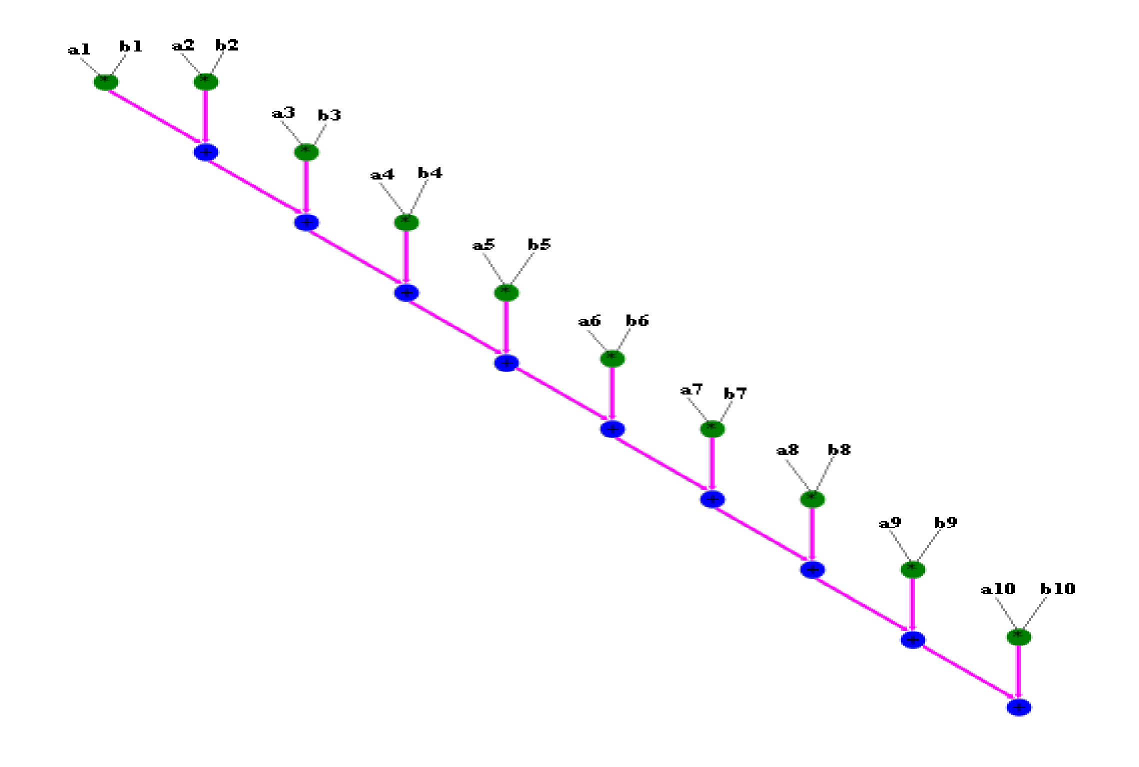
3- بعد الحصول على الصيغة Postfix لمعادلة الدخل سنستخدم خوارزمية DFG للحصول على اللائحة المترابطة التالية الموضحة في الجدول(4-1) التي تعبر عن مخطط تدفق المعطيات DFG:

الجدول (4-1) اللائحة المترابطة المعبرة عن DFG

Rec Num	Var left	Next left	opration	Next right	Var right
1	a1	Null	*	Null	b1
2	a2	Null	*	Null	b2
3		1	+	2	
4	a3	Null	*	Null	b3
5		3	+	4	
6	a4	Null	*	Null	b4
7		5	+	6	
8	a5	Null	*	Null	b5
9		7	+	8	
10	a6	Null	*	Null	b6
11		9	+	10	
12	a7	Null	*	Null	b7
13		11	+	12	

14	a8	Null	*	Null	b8
15		13	+	14	
16	a9	Null	*	Null	b9
17		15	+	16	
18	a10	Null	*	Null	b10
19		17	+	18	

4- من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية التالية الموضحة بالشكل (4-3) وذلك بدون تحديد الخطوات الزمنية للعمليات الحسابية:



الشكل (4-3) الشجرة الثنائية DFG

5- الخطوة التالية الانتقال من مخطط انسياب المعطيات DFG إلى مخطط انسياب المعطيات المجدول SDFG بدون قيود حيث تم إضافة الحقول التالية إلى اللائحة السابقة:

- حقل الـ Control-Step الذي يحدد لنا الخطوة الزمنية لتنفيذ العملية الحسابية.

- حقل الـ Level الذي يعطينا بُعد العملية الحسابية عن معطيات الدخل الأساسية.

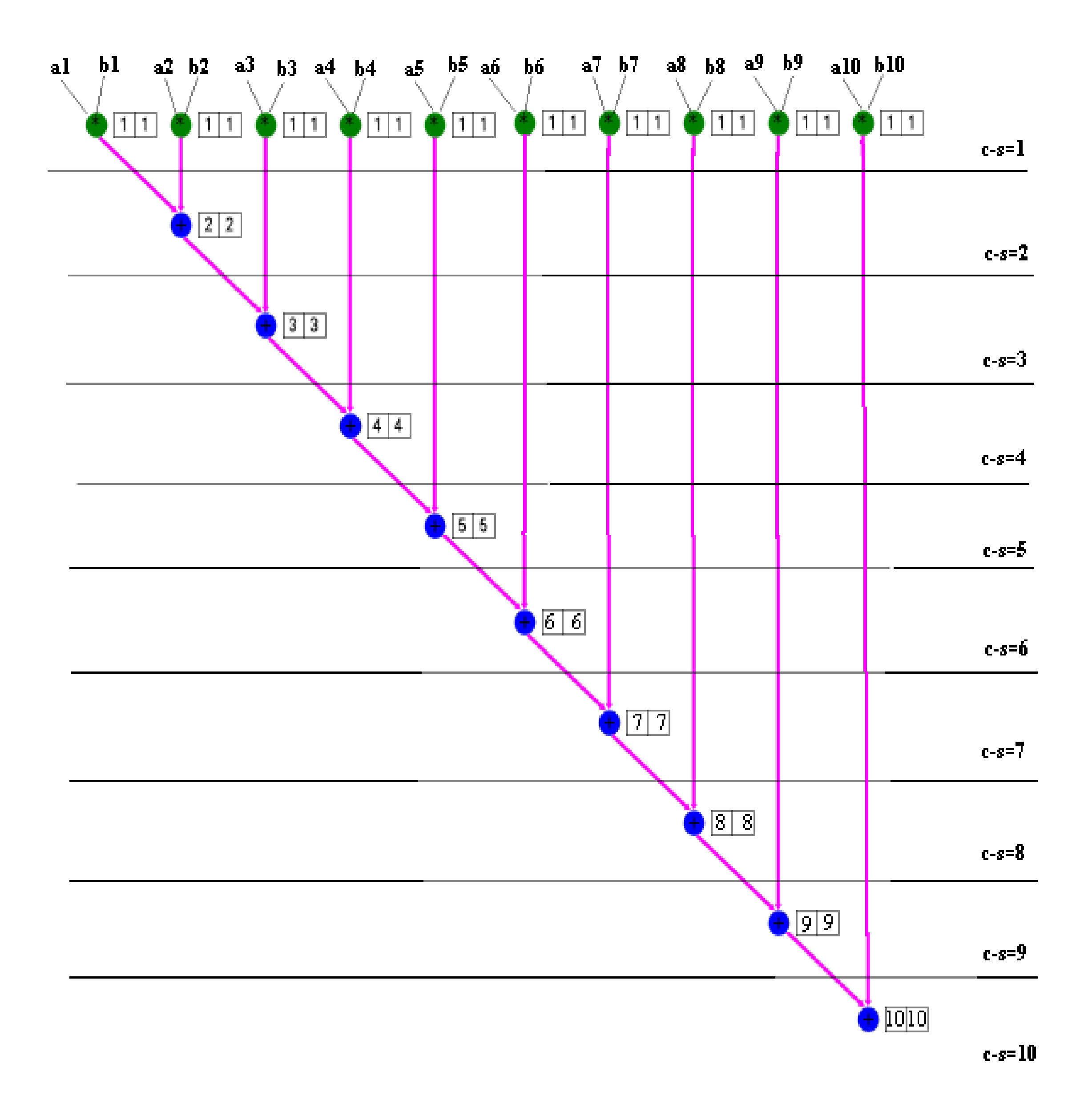
سنقوم باستخدام ثلاث خوارزميات للحصول على مخطط انسياب المعطيات المجدول بدون قيود وهي:

أ- خوارزمية ASAP بدون قيود ومنها نحصل على اللائحة المترابطة التالية الموضحة في الجدول (2-4):

الجدول (4-2) اللائحة المترابطة المعبرة عن SDFG باستخدام ASAP بدون قيود

Rec Num	Var left	Next left	opration	Next right	Var right	Level	C-Step
1	a1	Null	*	Null	b1	1	1
2	a2	Null	*	Null	b2	1	1
3		1	+	2		2	2
4	a3	Null	*	Null	b3	1	1
5		3	+	4		3	3
6	a4	Null	*	Null	b4	1	1
7		5	+	6		4	4
8	a5	Null	*	Null	b5	1	1
9		7	+	8		5	5
10	a6	Null	*	Null	b6	1	1
11		9	+	10		6	6
12	a7	Null	*	Null	b7	1	1
13		11	+	12		7	7
14	a8	Null	*	Null	b8	1	1
15		13	+	14		8	8
16	a9	Null	*	Null	b9	1	1
17		15	+	16		9	9
18	a10	Null	*	Null	b10	1	1
19		17	+	18		10	10

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية الموضحة في الشكل (4-4) حيث يتم فيها تحديد الخطوات الزمنية للعمليات الحسابية بطريقة ASAP بدون قيود:



الشكل (4-4) الشجرة الثنائية ASAP بدون قيود

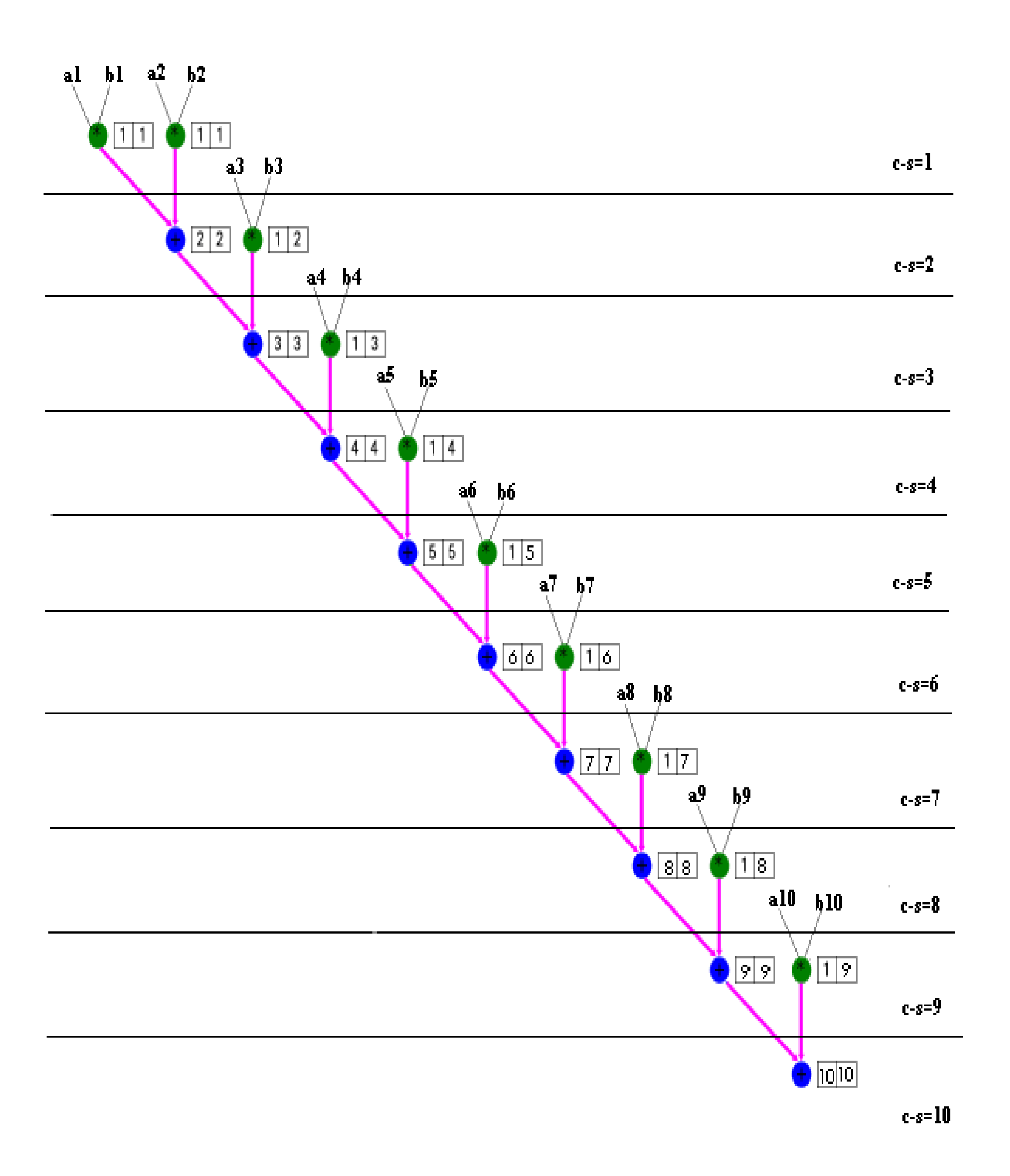
ب- خوارزمية ALAP بدون قيود ومنها نحصل على اللائحة المترابطة الموضحة في الجدول (3-4):

الجدول (3-4) اللائحة المترابطة المعبرة عن SDFG باستخدام ALAP بدون قيود

Rec Num	Var left	Next left	opration	Next right	Var right	Level	C-Step
1	a1	Null	*	Null	b1	1	1

2	a2	Null	*	Null	b2	1	1
3		1	+	2		2	2
4	a3	Null	*	Null	b3	1	2
5		3	+	4		3	3
6	a4	Null	*	Null	b4	1	3
7		5	+	6		4	4
8	a5	Null	*	Null	b5	1	4
9		7	+	8		5	5
10	a6	Null	*	Null	b6	1	5
11		9	+	10		6	6
12	a7	Null	*	Null	b7	1	6
13		11	+	12		7	7
14	a8	Null	*	Null	b8	1	7
15		13	+	14		8	8
16	a9	Null	*	Null	b9	1	8
17		15	+	16		9	9
18	a10	Null	*	Null	b10	1	9
19		17	+	18		10	10

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية الموضحة في الشكل (4–5) حيث يتم فيها تحديد الخطوات الزمنية للعمليات الحسابية بطريقة ALAP بدون قيود:



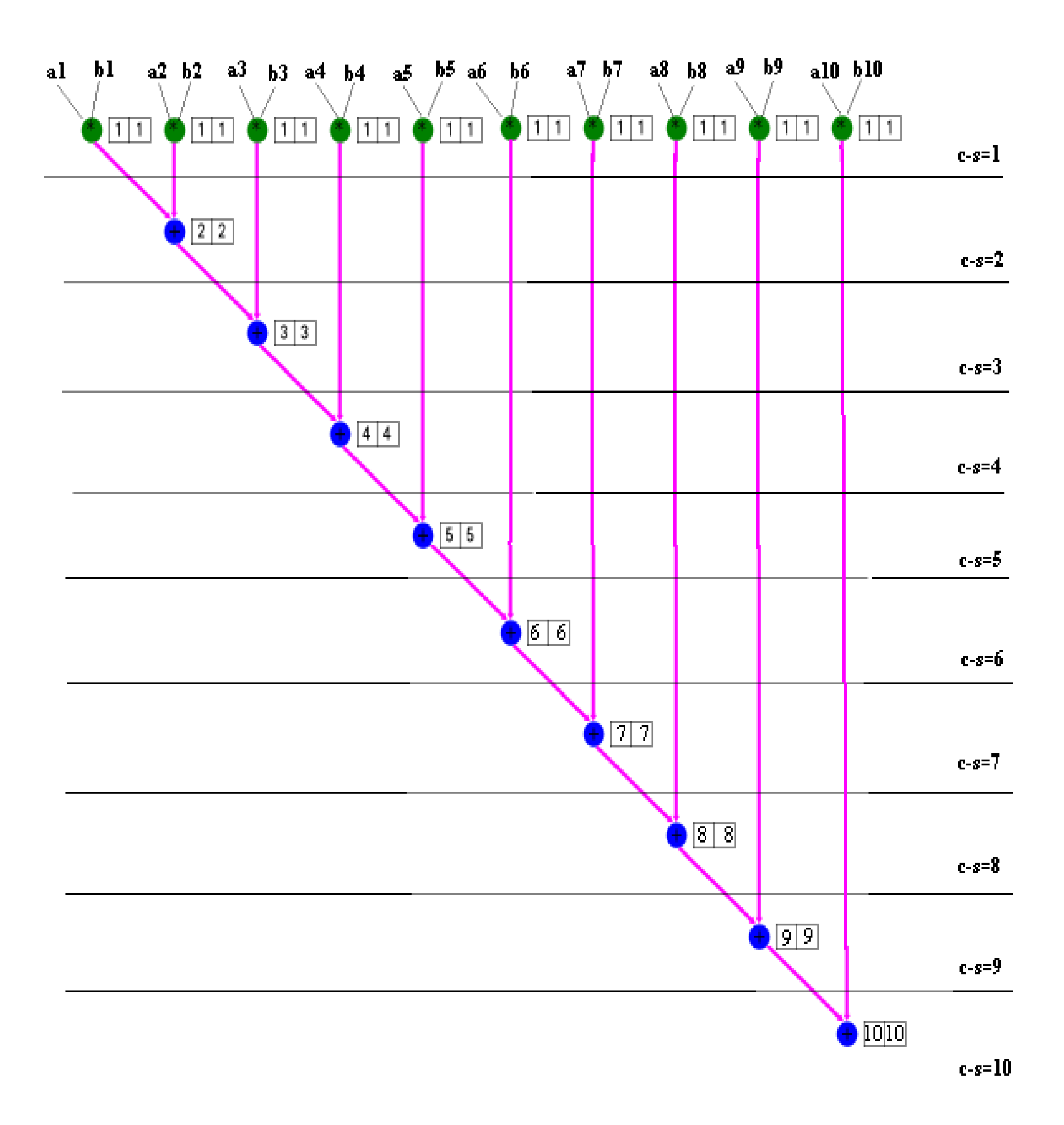
الشكل (4-5) الشجرة الثنائية ALAP بدون قيود

ج- خوارزمية List بدون قيود نحصل على اللائحة المترابطة الموضحة في الجدول(4-4):

الجدول(4-4) اللائحة المترابطة المعبرة عن SDFG باستخدام LIST بدون قيود

Rec Num	Var left	Next left	opration	Next right	Var right	Level	C-Step
1	a1	Null	*	Null	b1	1	1
2	a2	Null	*	Null	b2	1	1
3		1	+	2		2	2
4	a3	Null	*	Null	b3	1	2
5		3	+	4		3	3
6	a4	Null	*	Null	b4	1	3
7		5	+	6		4	4
8	a5	Null	*	Null	b5	1	4
9		7	+	8		5	5
10	a6	Null	*	Null	b6	1	5
11		9	+	10		6	6
12	a7	Null	*	Null	b7	1	6
13		11	+	12		7	7
14	a8	Null	*	Null	b8	1	7
15		13	+	14		8	8
16	a9	Null	*	Null	b9	1	8
17		15	+	16		9	9
18	a10	Null	*	Null	b10	1	9
19		17	+	18		10	10

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية الموضحة في الشكل (4-6) حيث يتم فيها تحديد الخطوات الزمنية للعمليات الحسابية بطريقة LIST بدون قيود:



الشكل (4-6) الشجرة الثنائية List بدون قيود

6- الخطوة التالية الانتقال من مخطط انسياب المعطيات المجدول SDFG بدون قيود إلى مخطط انسياب المعطيات المجدول SDFG مع قيود حيث تم إضافة الحقل التالي إلى اللائحة السابقة:

- حقل درجة الحرية Free_Degree الذي يعبر عن قابلية حركة العملية بين الخطوات الزمنية المختلفة:

ويحتسب بالعلاقة التالية وذلك بعد جدولة العملية بطريقتين ALAP & ASAP:

Free_D=[C_S(ALAP)]-[C_S(ASAP)]

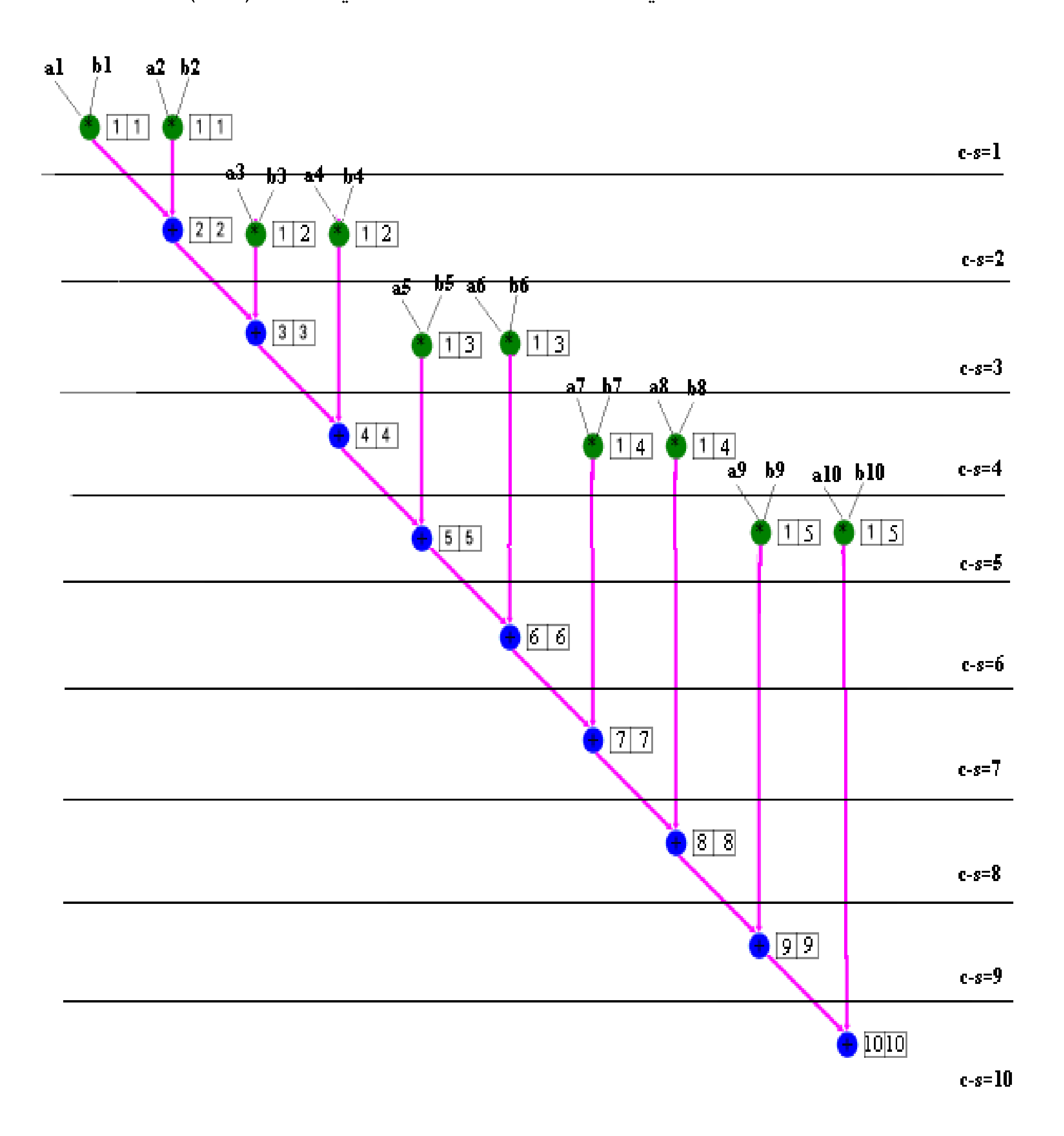
سنقوم باستخدام ثلاث خوارزميات للحصول على مخطط انسياب المعطيات المجدول مع قيود وهي:

أ- خوارزمية ASAP مع قيود ومنها نحصل على اللائحة المترابطة التالية الموضحة في الجدول(4-5) وذلك بفرض وجود قيد لا يسمح بأكثر من عمليتي ضرب في كل خطوة زمنية (لا نستطيع وضع قيود على عدد الجوامع بالنسبة لهذه العلاقة لأنه لايوجد أكثر من عملية جمع واحدة في كل خطوة زمنية):

الجدول(4-5) اللائحة المترابطة ASAP بقيود

Rec Num	Var left	Next left	opration	Next right	Var right	Level	C-Step	Free-D
1	a1	Null	*	Null	b1	1	1	0
2	a2	Null	*	Null	b2	1	1	0
3		1	+	2		2	2	0
4	a3	Null	*	Null	b3	1	2	1
5		3	+	4		3	3	0
6	a4	Null	*	Null	b4	1	2	2
7		5	+	6		4	4	0
8	a5	Null	*	Null	b5	1	3	3
9		7	+	8		5	5	0
10	a6	Null	*	Null	b6	1	3	4
11		9	+	10		6	6	0
12	a7	Null	*	Null	b7	1	4	5
13		11	+	12		7	7	0
14	a8	Null	*	Null	b8	1	4	6
15		13	+	14		8	8	0
16	a9	Null	*	Null	b9	1	5	7
17		15	+	16		9	9	0
18	a10	Null	*	Null	b10	1	5	8
19		17	+	18		10	10	0

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية المعبرة عن ASAP مع قيد مفروض على عدد الضوارب بحيث لا تتجاوز عمليتين في كل خطوة زمنية الموضحة في الشكل (4–7):



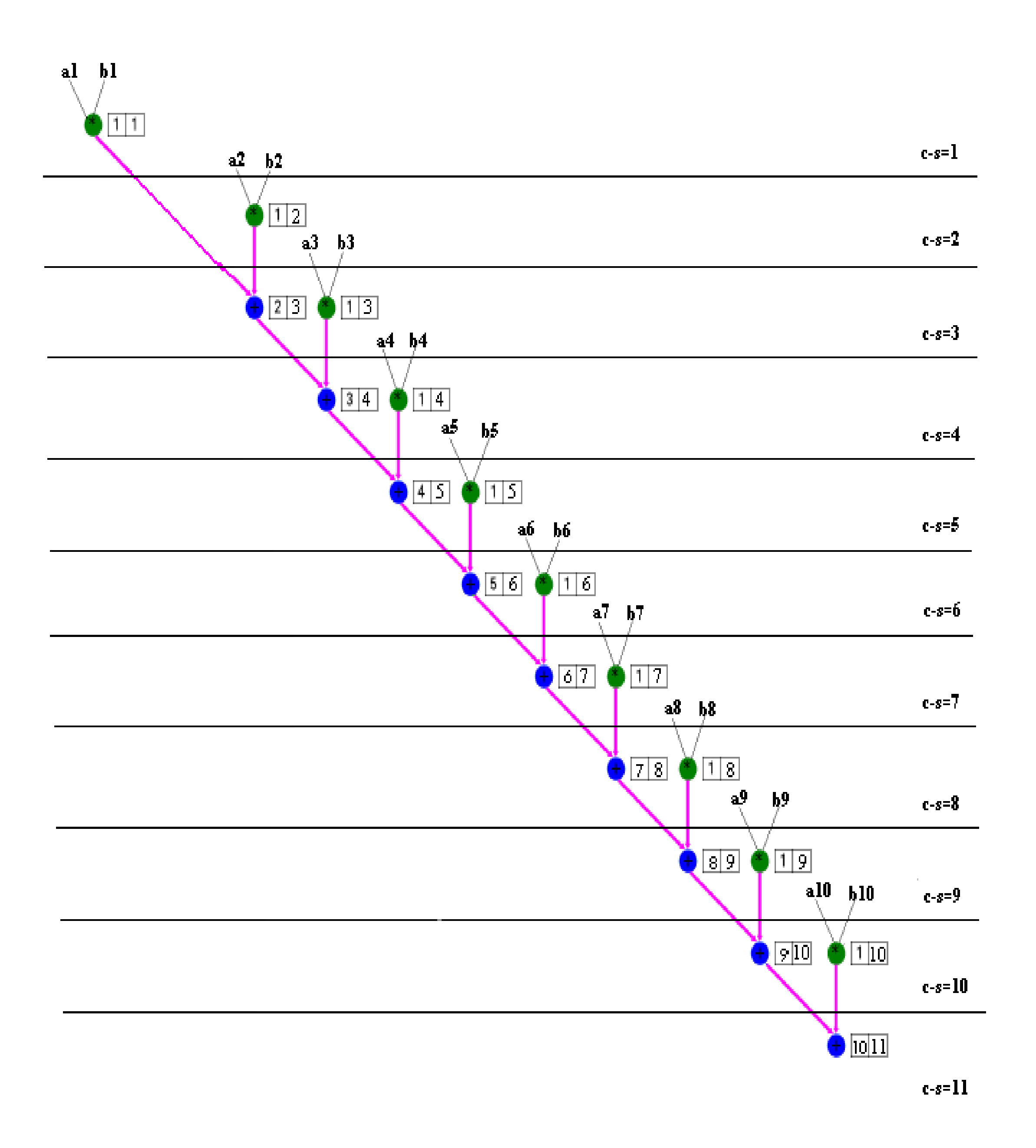
الشكل(4–7) الشجرة الثنائية ASAP بقيود

ب–خوارزمية ALAP بقيود ، إن القيد الوحيد على عدد الضوارب بالنسبة لهذه العلاقة هو وجود ضارب واحد فقط في كل خطوة زمنية بينما لا يمكن تطبيق أي قيد على عدد الجوامع لأنه بطبيعة الحال لا يوجد سوى عملية جمع واحدة في كل خطوة زمنية ، ومنه نحصل على اللائحة المترابطة التالية الموضحة في الجدول(4–6):

الجدول(4-6) اللائحة المترابطة ALAP بقيود

Rec Num	Var left	Next left	opration	Next right	Var right	Level	C-Step	Free-D
1	a1	Null	*	Null	b1	1	1	0
2	a2	Null	*	Null	b2	1	2	0
3		1	+	2		2	3	0
4	a3	Null	*	Null	b3	1	3	1
5		3	+	4		3	4	0
6	a4	Null	*	Null	b4	1	4	2
7		5	+	6		4	5	0
8	a5	Null	*	Null	b5	1	5	3
9		7	+	8		5	6	0
10	a6	Null	*	Null	b6	1	6	4
11		9	+	10		6	7	0
12	a7	Null	*	Null	b7	1	7	5
13		11	+	12		7	8	0
14	a8	Null	*	Null	b8	1	8	6
15		13	+	14		8	9	0
16	a9	Null	*	Null	b9	1	9	7
17		15	+	16		9	10	0
18	a10	Null	*	Null	b10	1	10	8
19		17	+	18		10	11	0

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية المعبرة عن ALAP مع قيد مفروض على عدد الضوارب بحيث لا تتجاوز عملية واحدة في كل خطوة زمنية الموضحة في الشكل (4-8):



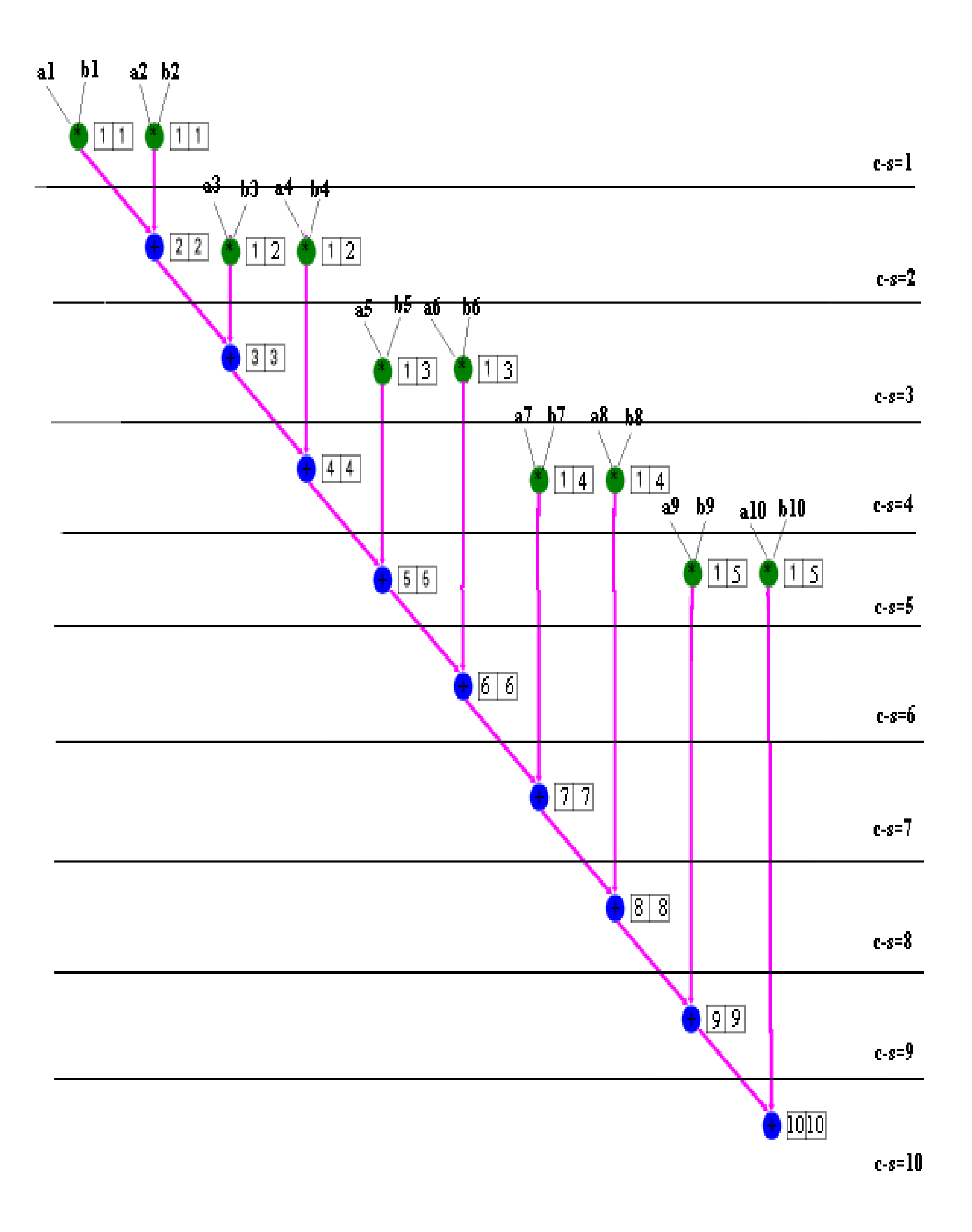
الشكل(4-8) الشجرة الثنائية ALAP بقيود

ت-خوارزمية List مع قيود ومنها نحصل على اللائحة المترابطة التالية الموضحة في الجدول(4-7) وذلك بفرض وجود قيد لا يسمح بأكثر من عمليتي ضرب في كل خطوة زمنية (سنستخدم رتل الأولويات لوضع العمليات فيه بحسب أفضلية تنفيذها في الأداة):

الجدول(4−7) اللائحة المترابطة List بقيود

Rec Num	Var left	Next left	opration	Next right	Var right	Level	C-Step
1	a1	Null	*	Null	b1	1	1
2	a2	Null	*	Null	b2	1	1
3		1	+	2		2	2
4	a3	Null	*	Null	b3	1	2
5		3	+	4		3	3
6	a4	Null	*	Null	b4	1	2
7		5	+	6		4	4
8	a5	Null	*	Null	b5	1	3
9		7	+	8		5	5
10	a6	Null	*	Null	b6	1	3
11		9	+	10		6	6
12	a7	Null	*	Null	b7	1	4
13		11	+	12		7	7
14	a8	Null	*	Null	b8	1	4
15		13	+	14		8	8
16	a9	Null	*	Null	b9	1	5
17		15	+	16		9	9
18	a10	Null	*	Null	b10	1	5
19		17	+	18		10	10

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية المعبرة عن List مع قيد مفروض على عدد الضوارب بحيث لا تتجاوز عمليتين في كل خطوة زمنية الموضحة في الشكل (4−9):



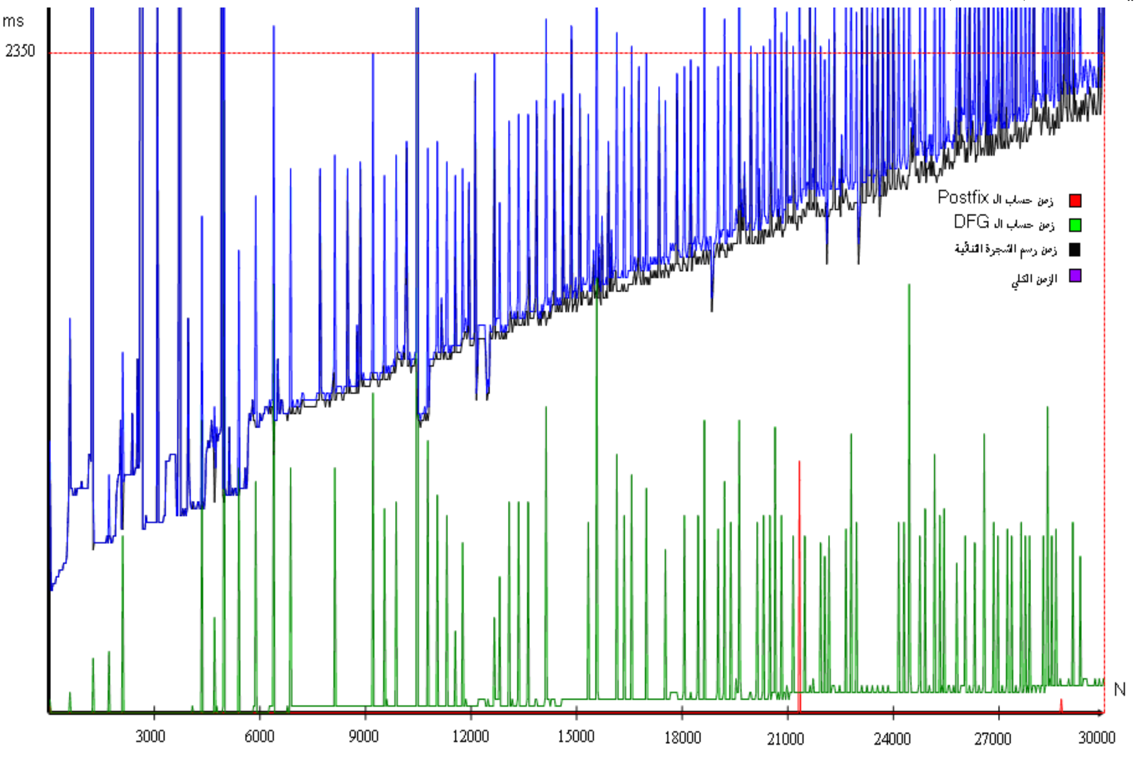
الشكل(4-9) الشجرة الثنائية List بقيود

4-3- النتائج الزمنية للبرامج باللغات الثلاث:

في كل مرحلة من المراحل السابقة تم حساب زمن التنفيذ وذلك بالنسبة للغات البرمجة الثلاثة (Java – Delphi – C#).

4-3-1- النتائج الزمنية لبرنامج الـ C# :

عند تشغيل نسخة البرنامج المكتوبة بلغة الـ C# حصلنا على المخطط البياني التالي الموضح في الشكل (4-10).



الشكل (4-10) المخطط الزمني باستخدام لغة C#

نلاحظ من هذا المخطط أن هناك تزايد بشكل تصاعدي تدريجي خطي في الزمن مع ازدياد قيمة N تتخللها قفزات هائلة في الزمن ليعود مرة أخرى إلى حدوده الطبيعية وذلك بسبب انشغال المعالج بأعمال أخرى تتطلبها بيئة الويندوز وهوما يعرف بالـ Thread، بشكل عام البرنامج يعطي نتائج جيدة من حيث زمن التنفيذ وكمية المعطيات المعالجة بالإضافة إلى جودة في رسم الأشجار الثنائية وسهولة في التعامل مع عناصر الرسم بسبب غناها بالمكتبات الداعمة للرسوميات، كما أنه باستخدام لغة C# هناك إمكانية في الوصول إلى قيم كبيرة لمعادلة الدخل تصل إلى N=500000 أو أكثر.

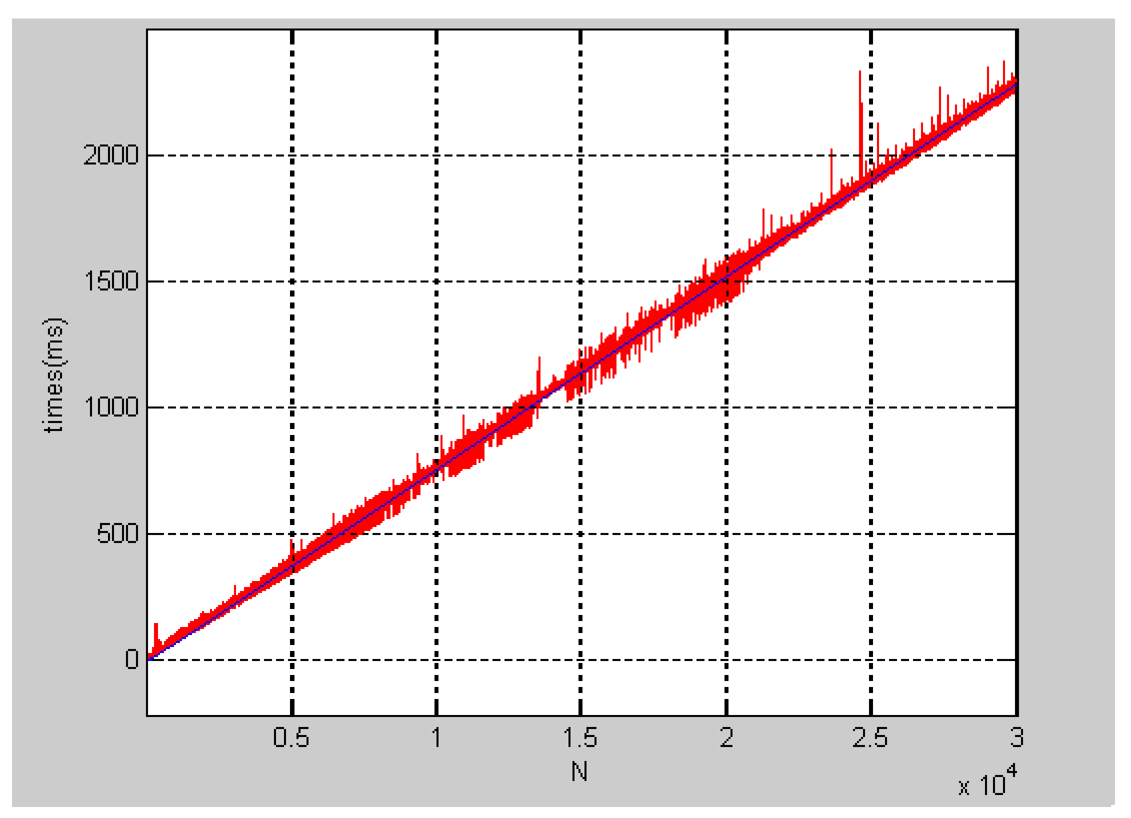
باستخدام برنامج MATLAB نستطيع الحصول على معادلة خطية (كثير حدود من الدرجة الأولى) مارة بالمنحني السابق وذلك حتى نسبة خطأ من مرتبة (1e-008) ،حيث عند هذه النسبة تصبح المعادلة معادلة كثير حدود من الدرجة الثانية ولكن بأمثال صغيرة جداً لـ X^2 من مرتبة (1e-008) وبالتالي يمكن إهمالها ، إن هذه النتيجة تم التوصل إليها بالانتقال التدريجي من نسبة خطأ من مرتبة 1e-003 حتى وصلنا إلى النسبة المطلوبة كما هوموضح في الجدول(4-8) :

الجدول(4-8) يبين معادلة المنحني المار بالمخطط الزمني من أجل نسب خطأ مختلفة

نسبة الخطأ	معادلة كثير الحدود
1e-003	0.076153 x - 3.63
1e-004	0.076153 x - 3.63
1e-005	0.076153 x - 3.63
1e-006	0.076153 x - 3.63
1e-007	0.076153 x - 3.63
1e-008	-1.0196e-008 x^2 + 0.076458 x - 5.1596

من الجدول السابق نلاحظ أن معادلة المنحني المار بنقاط المخطط الزمني السابق النهائية هي من الشكل الخطي Ax+B مع نسبة خطأ مقدارها (|-1.0196e-008|) .

ومنه يمكن تمثيل المخطط الزمني السابق كما هوموضح بالشكل(4-11):



الشكل(4-11) المنحني المار بالمخطط الزمني باستخدام لغة #C

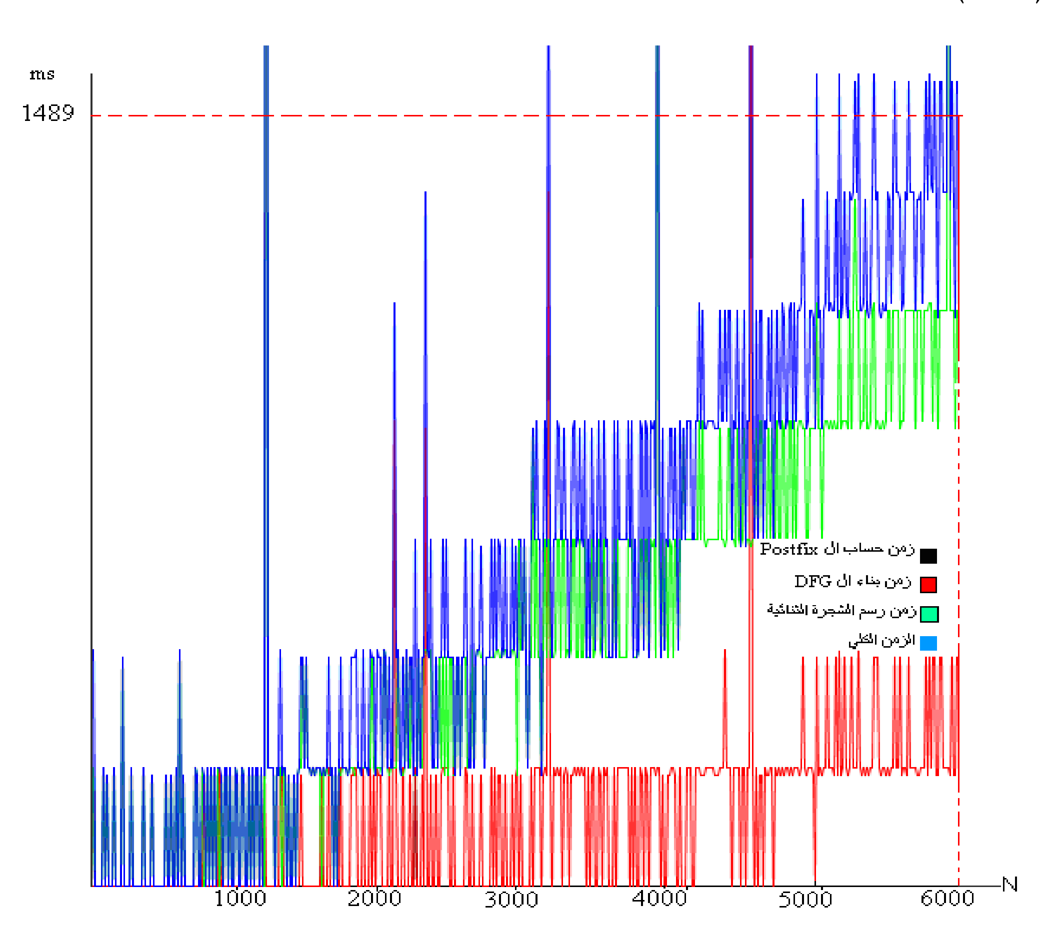
أما بالنسبة لحيز التخزين في لغة #C فقد تم حسابه كما يلي:

وجدنا أن بنية السجل للائحة المترابطة SDFG الموضحة في الشكل(3-8) تحتوي على ثلاثة حقول من نوع Charوالتي يُحجز لها في الذاكرة 1 Byte وخمسة حقول من نوع Integerالتي يُحجز

لها في الذاكرة 4Byte ليكون حجم الذاكرةالتي تُحجز للسجل الواحد هو 23Byte ، وبالتالي يكون حجم الذاكرة المحجوزة للائحة ككل من أجل علاقة بطول I هو [23*(2I-1)Byte] ، أما من أجل N علاقة تبدأ من I=1..N يكون حجم الذاكرة المحجوزة هو [23*N2 Byte] .

4-3-2- النتائج الزمنية لبرنامج الـ Java :

عند تشغيل نسخة البرنامج المكتوبة بلغة Java حصلنا على المخطط البياني التالي الموضح في الشكل (4-12).



الشكل (4-12) المخطط الزمني باستخدام لغة Java

نلاحظ من هذا المخطط أنه من أجل قيم صغيرة لـ N هناك سرعة كبيرة في التنفيذ تتباطأ مع ازدياد قيمة Nلتصل إلى سرعة بطيئة جداً مقارنةً مع لغة #Cوذلك اعتباراً من N= 1500 تقريباً ، كما نلاحظ وجود فترات من الثبات في أزمنة التنفيذ ضمن مجال من قيم N لتزداد فجأة بشكل كبير ضمن المجال التالي وذلك يعود إلى ازدياد حجم الذاكرة المستخدمة بشكل كبير ، أما بالنسبة لجودة رسم

الأشجار الثنائية فهي جيدة كما أن التعامل مع عناصر الرسم يتمتع بالسهولة نسبياً في هذه اللغة بسبب غناها أيضاً بالمكتبات الداعمة للرسوميات.

بالإضافة إلى ذلك استطعنا باستخدام لغة Java التوصل لقيم أكبر من N=100000 ولكن بزمن تنفيذ أكبر بكثير من لغة #C .

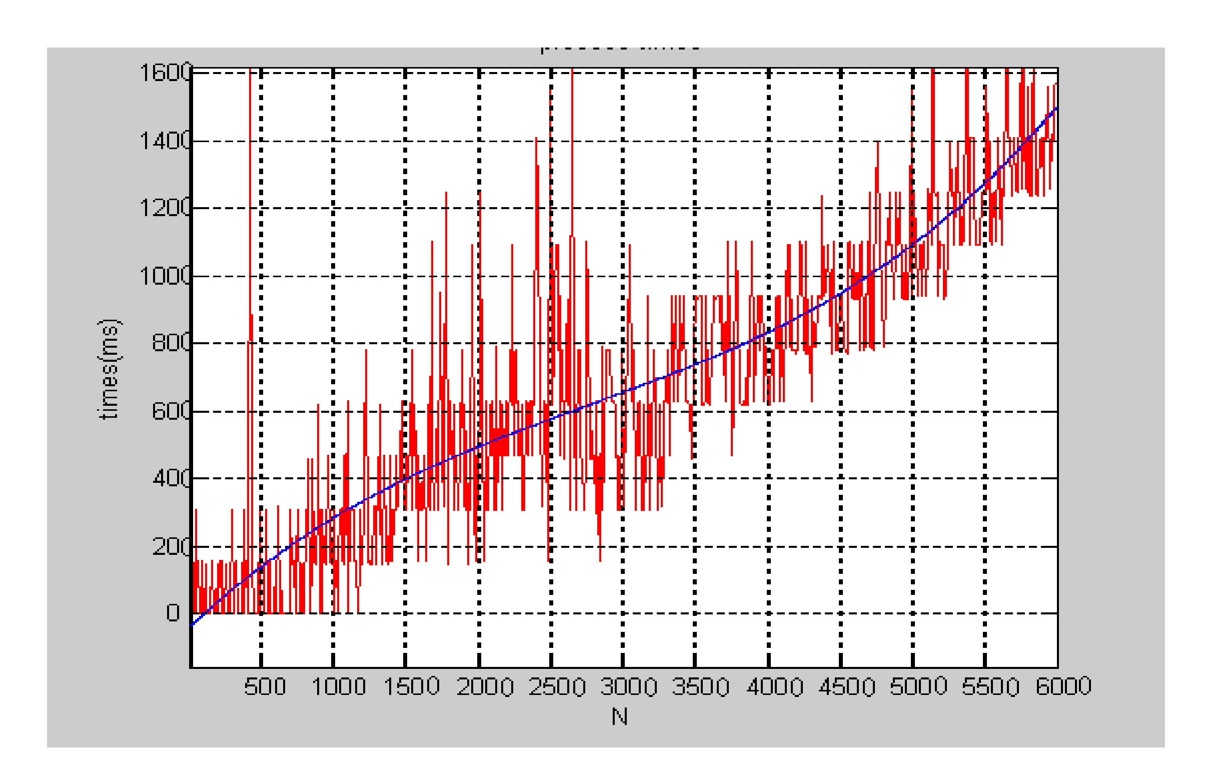
باستخدام برنامج MATLAB نستطيع الحصول على معادلة كثير حدود من الدرجة الثانية مارة بالمنحني السابق وذلك حتى نسبة خطأ من مرتبة (1e-006) ،حيث عند هذه النسبة تصبح المعادلة معادلة كثير حدود من الدرجة الثالثة ولكن بأمثال صغيرة جداً لـ X^3 من مرتبة (1e-006) وبالتالي يمكن إهمالها ، إن هذه النتيجة تم التوصل إليها بالانتقال التدريجي من نسبة خطأ من مرتبة 1e-003 حتى وصلنا إلى النسبة المطلوبة كما هوموضح في الجدول(4–9) :

الجدول(4–9) يبين معادلة المنحني المار بالمخطط الزمني من أجل نسب خطأ مختلفة

نسبة الخطأ	معادلة كثير الحدود
1e-003	0.21702 x + 2.9639
1e-004	8.3417e-005 x^2 + 0.16689 x + 7.994
1e-005	8.3417e-005 x^2 + 0.16689 x + 7.994
1e-006	1.1e-006 x^3 - 0.0009082 x^2 + 0.40547 x - 4.0048
1e-007	1.1e-006 x^3 - 0.0009082 x^2 + 0.40547 x - 4.0048
1e-008	1.1e-006 x^3 - 0.0009082 x^2 + 0.40547 x - 4.0048

من الجدول السابق نلاحظ أن معادلة المنحني المار بنقاط المخطط الزمني السابق النهائية هي من معادلة كثير حدود من الدرجة الثانية Ax^2+Bx+C مع نسبة خطأ مقدارها (|1.1e-006|) .

ومنه يمكن تمثيل المخطط الزمني السابق كما هوموضح بالشكل(4–13):



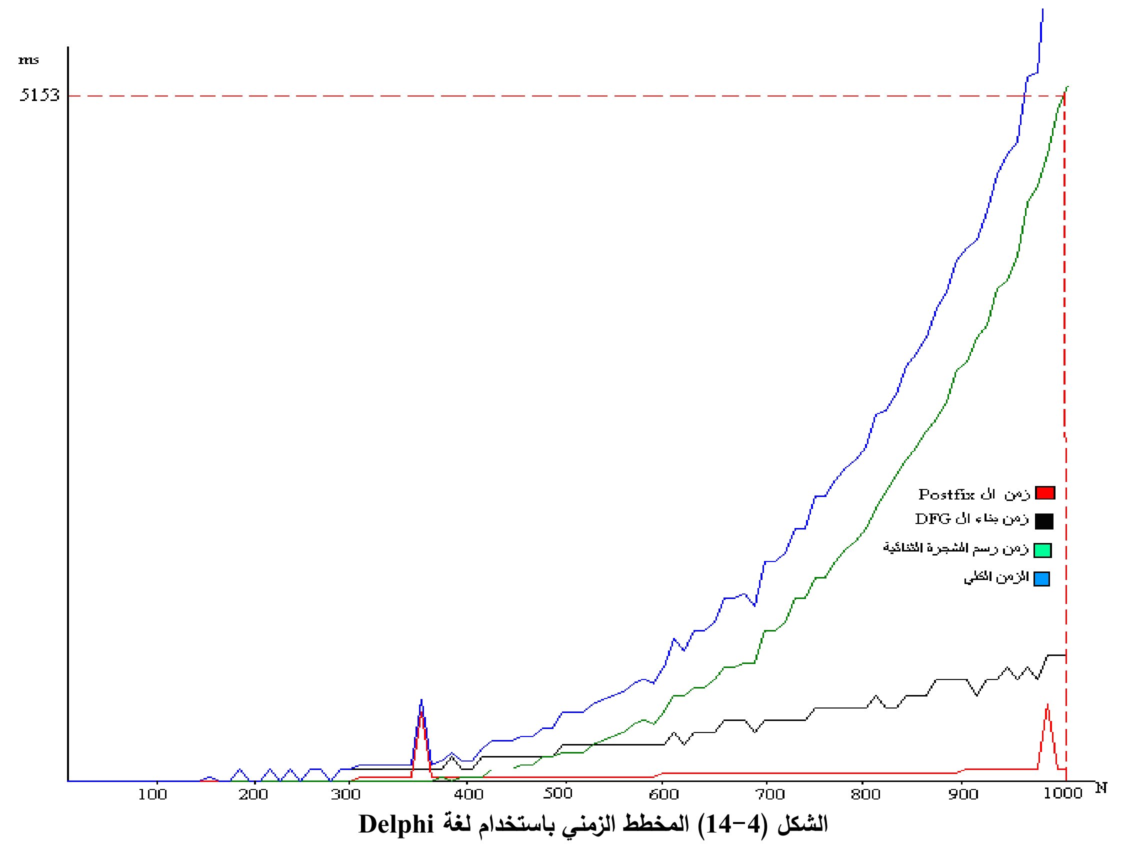
الشكل(4-13) المنحني المار بالمخطط الزمني باستخدام لغة Java

أما بالنسبة لحيز التخزين في لغة Java فقد تم حسابه كما يلي:

وجدنا أن بنية السجل للائحة المترابطة SDFG الموضحة في الشكل(3-8) تحتوي على ثلاثة حقول من نوع Charوالتي يُحجز لها في الذاكرة 1 Byte وخمسة حقول من نوع Integerالتي يُحجز لها في الذاكرة 4Byteليكون حجم الذاكرة التي تُحجز للسجل الواحد هو 23Byte ، وبالتالي يكون حجم الذاكرة المحجوزة للائحة ككل من أجل علاقة بطول I هو [23*(2I-1)Byte] ، أما من أجل N علاقة تبدأ من I=1..N يكون حجم الذاكرة المحجوزة هو [23*N2 Byte] .

4-3-3- النتائج الزمنية لبرنامج الـ Delphi :

أما عند تشغيل نسخة البرنامج المكتوبة بلغة Dephi حصلنا على المخطط البياني التالي الموضح في الشكل (4-14).



الشكل (4-14) المخطط الزمني باستخدام لغة Delphi

نلاحظ من هذا المخطط البياني أن الزمن يزداد بشكل أسي كلما زادت قيمة N حيث تبدأ الزيادة الكبيرة في الزمن من أجل N>500. أما بالنسبة لجودة رسم الأشجار الثنائية فهي ضعيفة حيث هناك صعوبة كبيرة في التعامل مع عناصر الرسم بسبب عدم توفر المكتبات الداعمة للرسوميات.

بالإضافة إلى ذلك لم نستطع باستخدام لغة Delphi التوصل لقيم أكبر من N=1000 حيث لم يعطي البرنامج أي استجابة وذلك لأن الذاكرة المخصصة للبرنامج ليست بمستوى لغات C# و Java اللتين تحتويان على ذاكرة بحجم مخصص للتطبيقات الكبيرة .

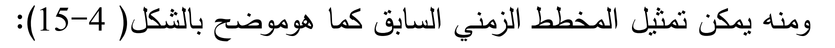
باستخدام برنامج MATLAB نستطيع الحصول على معادلة كثير حدود من الدرجة الرابعة مارة بالمنحني السابق وذلك حتى نسبة خطأ من مرتبة (1e-007) ،حيث عند هذه النسبة تصبح المعادلة معادلة كثير حدود من الدرجة الخامسة ولكن بأمثال صغيرة جداً لـ X^5 من مرتبة (1e-007) وبالتالي يمكن إهمالها ، إن هذه النتيجة تم التوصل إليها بالانتقال التدريجي من نسبة خطأ من مرتبة 1e-003 حتى وصلنا إلى النسبة المطلوبة كما هوموضح في الجدول(4-10) :

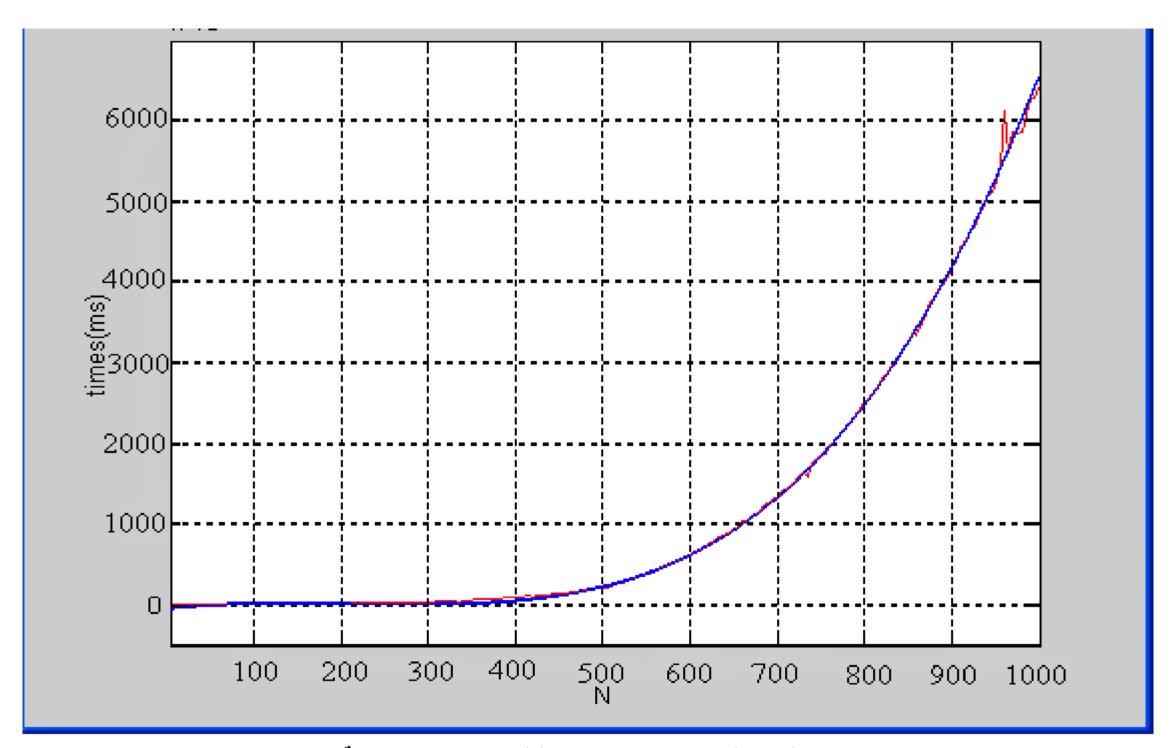
الجدول(4-10) يبين معادلة المنحني المار بالمخطط الزمني من أجل نسب خطأ مختلفة

نسبة الخطأ	معادلة كثير الحدود

1e-003	0.0098145 x^3 - 1.4551 x^2 + 63.2568 x - 591.3457
1e-004	0.0098145 x^3 - 1.4551 x^2 + 63.2568 x - 591.3457
1e-005	1.49e-005 x^4 + 0.0038248 x^3 - 0.67987 x^2 + 28.4265 x - 233.4492
1e-006	1.49e-005 x^4 + 0.0038248 x^3 - 0.67987 x^2 + 28.4265 x - 233.4492
1e-007	-1.9521e-007 x^5 + 0.00011299 x^4 - 0.013723 x^3 + 0.64783 x^2 - 10.0808 x + 33.5634
1e-008	-1.9521e-007 x^5 + 0.00011299 x^4 - 0.013723 x^3 + 0.64783 x^2 - 10.0808 x + 33.5634

من الجدول السابق نلاحظ أن معادلة المنحني المار بنقاط المخطط الزمني السابق النهائية هي معادلة كثير حدود من الدرجة الرابعة تأخذ الشكل Ax^4+Bx^3+Cx^2+Dx+E مع نسبة خطأ مقدارها (|-1.9521e-007|) .


ومنه يمكن تمثيل المخطط الزمني السابق كما هوموضح بالشكل(4-15):



الشكل(4-15) المنحني المار بالمخطط الزمني باستخدام لغة Delphi

أما بالنسبة لحيز التخزين في لغة Delphi فقد تم حسابه كما يلي:

وجدنا أن بنية السجل للائحة المترابطة SDFG الموضحة في الشكل(3-8) تحتوي على ثلاثة حقول من نوع Charوالتي يُحجز لها في الذاكرة 1 Byte وخمسة حقول من نوع Integerالتي يُحجز لها في الذاكرة 4Byteليكون حجم الذاكرة التي تُحجز للسجل الواحد هو 23Byte ، وبالتالي يكون

حجم الذاكرة المحجوزة للائحة ككل من أجل علاقة بطول I هو [23*(2I-1)Byte] ، أما من أجل N علاقة تبدأ من I=1..N يكون حجم الذاكرة المحجوزة هو [23*N2 Byte] .

4-4- الخلاصة:

يمكن تلخيص النتائج التي تم الحصول عليها في الجدول (4-11).

الجدول (4-11) نتائج المقارنة بين اللغات الثلاث

<table>
<tr><th>لغة البرمجة</th><th>سرعة التنفيذ</th><th>جودة رسم الأشجار الثنائية</th><th>كمية المعطيات المعالجة</th><th>معادلة المخطط الزمني</th></tr>
<tr><td>C#</td><td>تزايد بشكل تصاعدي تدريجي خطي في الزمن مع ازدياد قيمة N.</td><td>جودة عالية وسهولة في التعامل مع عناصر الرسم بسبب غناها بالمكتبات الداعمة للرسوميات.</td><td>إمكانية في الوصول إلى قيم كبيرة لمعادلة الدخل قد تصل إلىN=500000 أو أكثر.</td><td>كثير حدود من الدرجة الأو لى (خطي):
Ax+B
بخطأمقداره
|-1.0196e-008|</td></tr>
<tr><td>Java</td><td>من أجل قيم صغيرة لـ N هناك سرعة كبيرة في التنفيذ تتباطأ مع ازدياد قيمة N لتصل إلى سرعة بطيئة جداً مقارنةً مع لغة C#وذلك اعتباراً من N=1500 تقريباً.</td><td>جيدة كما أن التعامل مع عناصر الرسم يتمتع بالسهولة نسبياً في هذه اللغة بسبب غناها أيضاً بالمكتبات الداعمة للرسوميات.</td><td>إمكانية التوصل لقيم أكبر من N=100000 ولكن بأزمنة أبطأمن لغة C#.</td><td>كثير حدود من الدرجة الثانية:
Ax^2+Bx+C
بخطأمقداره
|1.1e-006|</td></tr>
<tr><td>Delphi</td><td>الزمن يزداد بشكل أسي كلما زادت قيمة N حيث تبدأ الزيادة الكبيرة في الزمن من أجل N>500.</td><td>ضعيفة حيث هناك صعوبة كبيرة في التعامل مع عناصر الرسم بسبب عدم توفر المكتبات الداعمة للرسوميات.</td><td>لا يمكن التوصل لقيم أكبر من N=1000 حيث لم يعطي البرنامج أي استجابة وذلك لكون الذاكرة المخصصة للتطبيقات صغيرة.</td><td>كثير حدود من الدرجة الرابعة:
Ax^4+Bx^3+Cx^2+Dx+E
بخطأمقداره
|-1.9521e-007|</td></tr>
</table>

مما تقدم نجد أن اللغة الأنسب لمتابعة العمل في بناء الأداة المنطقية هي لغة الـ C# لما تملكه من خصائص مميزة في التعامل مع عناصر الرسم بالإضافة إلى أزمنة التنفيذ الجيدة مقارنةً مع غيرها، وكمية المعطيات الكبيرة نسبياً القابلة للمعالجة.

الفصل الخامس

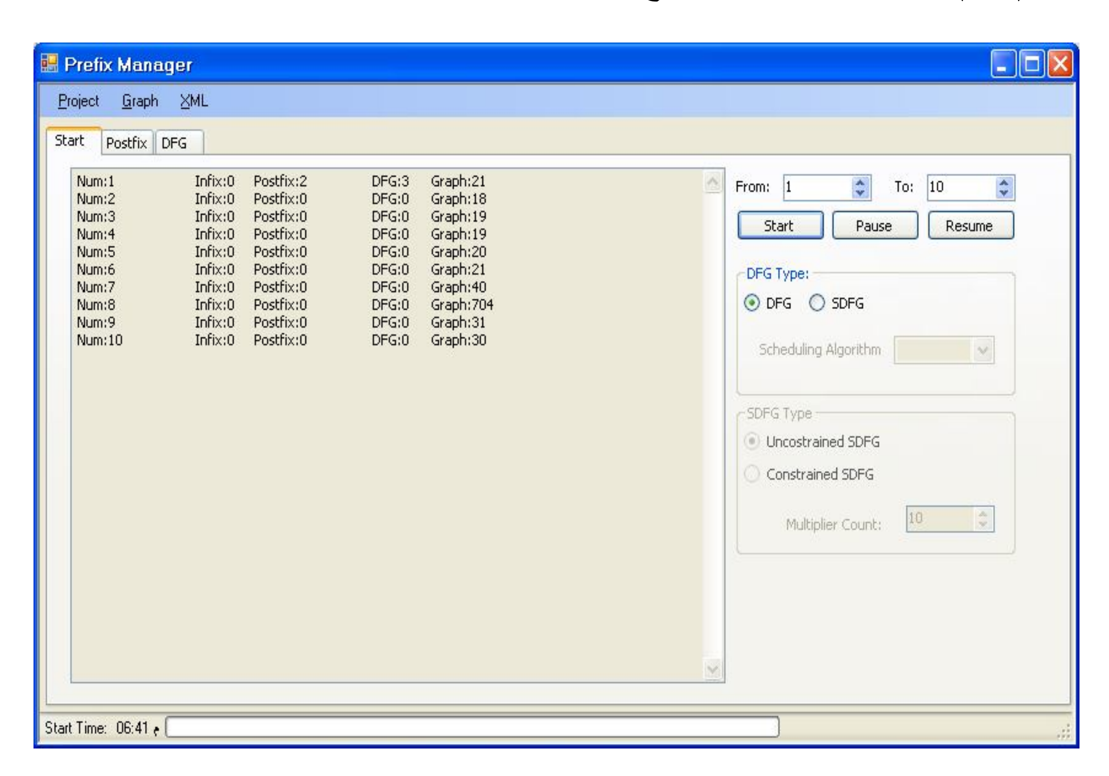
واجهات البرامج التي تم استخدامها في مرحلة المقارنة بين اللغات الثلاثة

في هذا الفصل سنستعرض العمل البرمجي للغات الثلاثة التي استخدمناها لإجراء المقارنة في الفصل السابق .

5-1- واجهات البرنامج المكتوب بلغة C# :

نستعرض فيما يلي واجهات البرنامج المكتوب بلغة الـ C# مع شرح مفصل لكل مكوناتها:

الشكل (5-1)يبين الواجهة الرئيسية لبرنامج الـ C# :



الشكل (5-1) الواجهة الرئيسية لبرنامج الـ C# .

تشتمل الواجهة الرئيسية على ثلاثة تبويبات وثلاثة قوائم ، ولنبدأ بشرح التبويبات :

1- التبويب الأول Start : في هذا التبويب نحدد القيمة البدائية والنهائية للعلاقة العامة للمرشحات. From: 1 To: 10

ثم نحدد المرحلة التي سيتم تطبيقها فيما إذا كانت مرحلة ترجمة أو مرحلة جدولة باستخدام إحدى خوارزميات الجدولة.



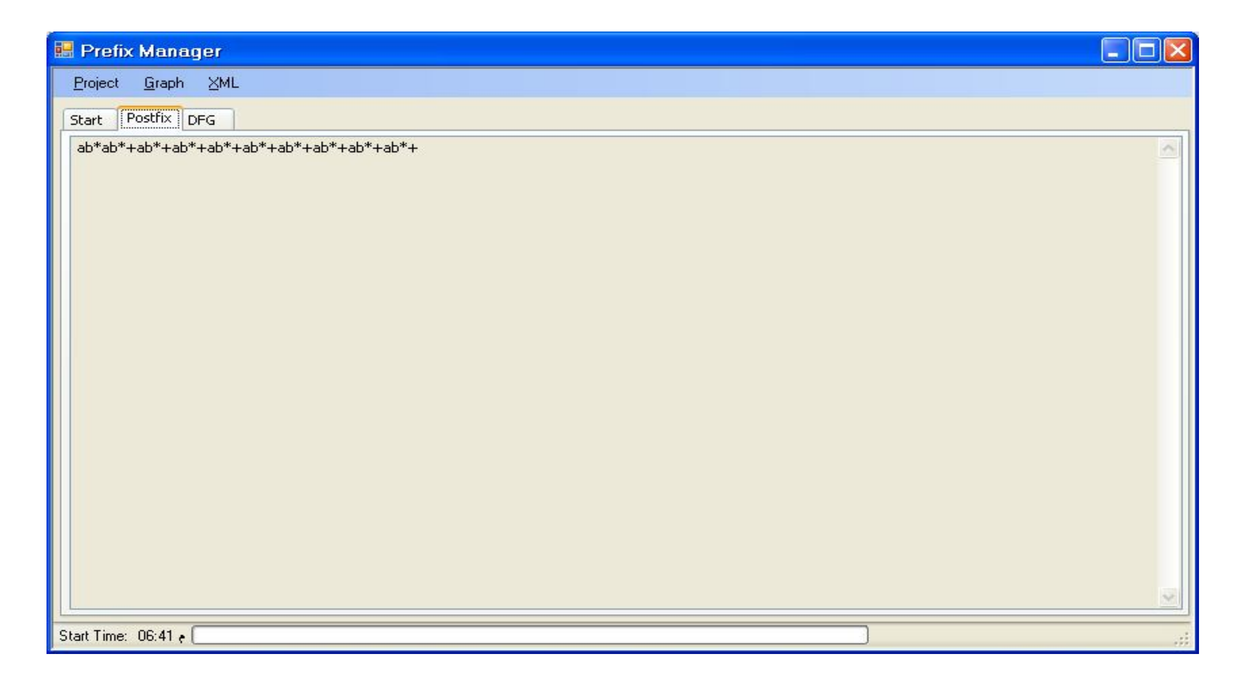
في حالة تم اختيار مرحلة الجدولة نحدد نوع الجدولة بقيود أو بدون قيود مفروضة على عدد الضوارب فقط وذلك لأنه لا يمكن فرض قيود على عدد الجوامع بالنسبة للعلاقة موضوع الدراسة .



يحتوي هذا التبويب أيضاً على ثلاثة أزرار هي زر Start للبدء بعملية التنفيذ اعتباراً من القيمة البدائية وحتى القيمة النهائية حيث يتم حساب الأزمنة اللازمة لمرحلتي الترجمة والجدولة ، أما بالنسبة لزر Pause فهو من أجل إيقاف عملية التنفيذ بشكل مؤقت ومن ثم متابعة التنفيذ من نقطة التوقف باستخدام زر Resume .



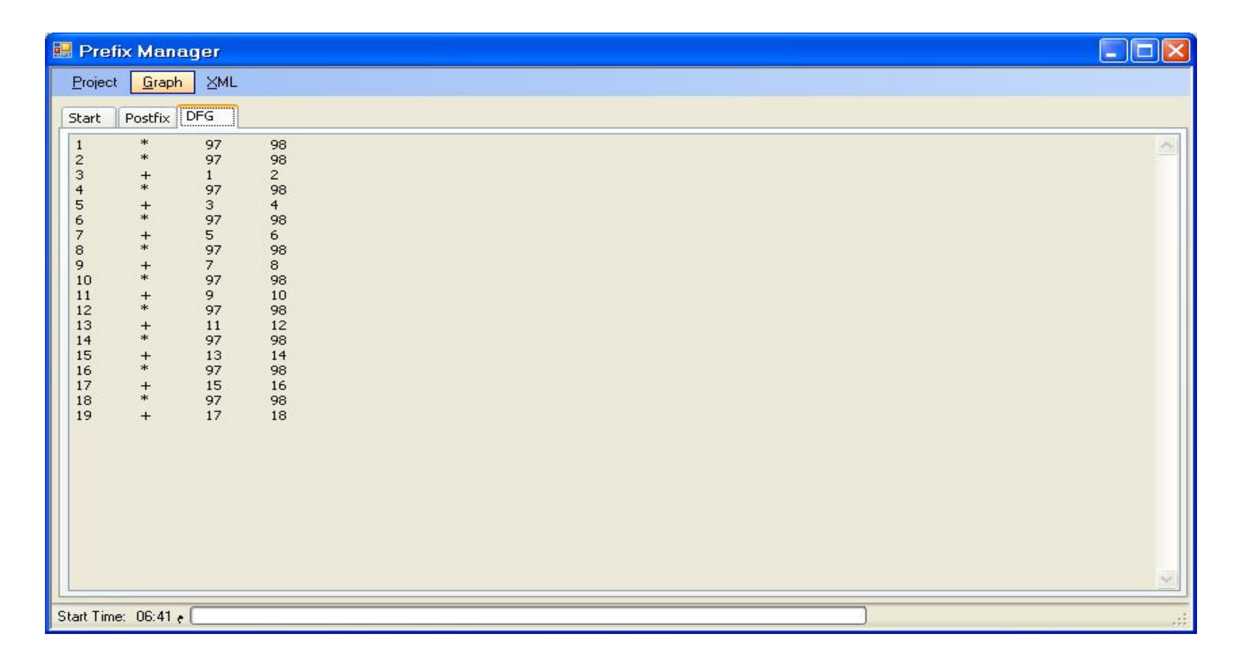
2- التبويب الثاني Postfix : في هذا التبويب يتم إظهار صيغة الـ Postfix للعلاقة بقيمتها النهائية اعتماداً على خوارزمية التحويل من Infix إلى Postfix كما هو موضح في الشكل (2-5):



الشكل (5-2) التبويب الثاني للواجهة الرئيسية لبرنامج الـ #C .

3- التبويب الثالث DFG : في هذا التبويب يتم إظهار جدول الـ DFG أو جدول الـ SDFG للعلاقة بقيمتها النهائية حسب المرحلة المختارة في التبويب الأول وذلك باستخدام خوارزميات الترجمة والجدولة المناسبة، ويتم تخزين الجدول الناتج في الذاكرة على شكل لائحة مترابطة.

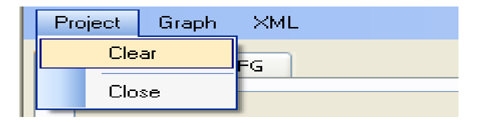
يبين الشكل (5-3) هذا التبويب:



الشكل (5-3) التبويب الثالث للواجهة الرئيسية لبرنامج الـ #C .

تحتوي الواجهة الرئيسية أيضاً على ثلاثة قوائم تحتوي على مجموعة من الأوامر وهي :

1- القائمة Project : موضحة في الشكل (5-4):

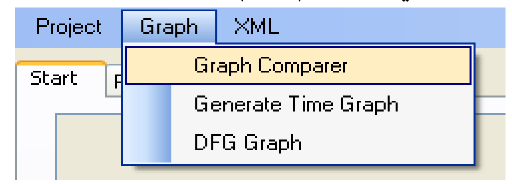


الشكل (5-4) القائمة Project للواجهة الرئيسية لبرنامج الـ C# .

تحتوى هذه القائمة على الأوامر التالية :

- الأمر Clear : يستخدم لمسح النتائج وتصفير الذاكرة لإتاحة المجال للبدء بتنفيذ جديد مع قيم جديدة.
- الأمر Close : يستخدم لإنهاء البرنامج والخروج منه .

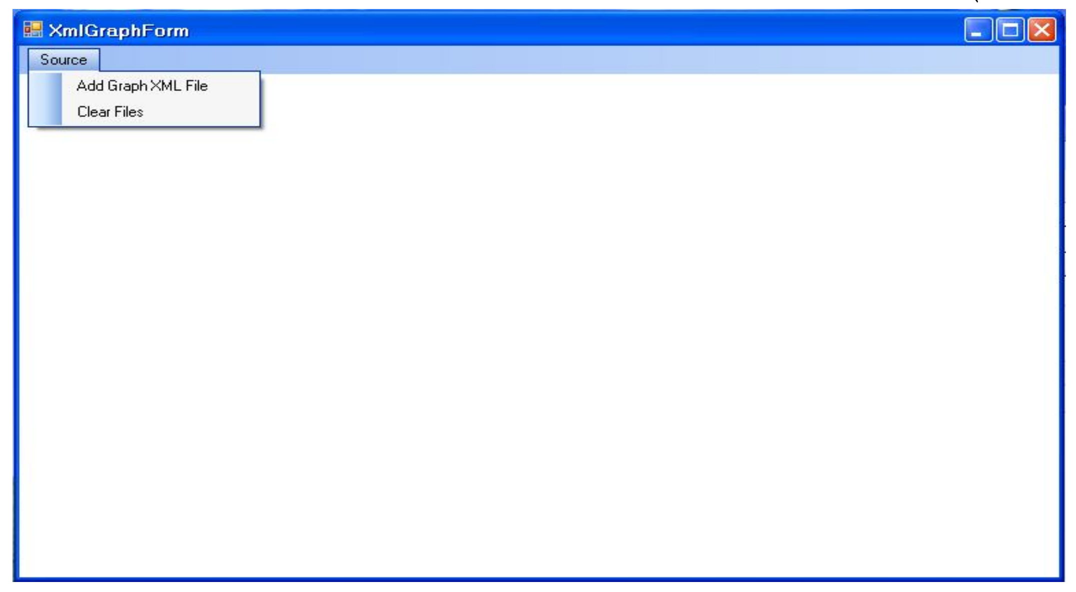
2- القائمة Graph : موضحة في الشكل (5-5):



الشكل (5-5) القائمة Graph للواجهة الرئيسية لبرنامج الـ C# .

تحتوى هذه القائمة على الأوامر التالية:

- الأمر Graph Comparer : بهذا الأمر ننتقل إلى الواجهة الثانوية الموضحة في الشكل (5-6):



الشكل (5-6) الواجهة الثانوية Graph Comparer لبرنامج الـ C# .

في هذه الواجهة توجد القائمة Source التي تحتوي على الأوامر التالية :

أ- الأمر Add Graph XML File : الذي يقوم باستيراد ملفات النتائج الزمنية المخزنة على شكل ملفات XML من أجل إجراء عمليات المقارنة بينها وذلك حسب الخوارزمية التالية:

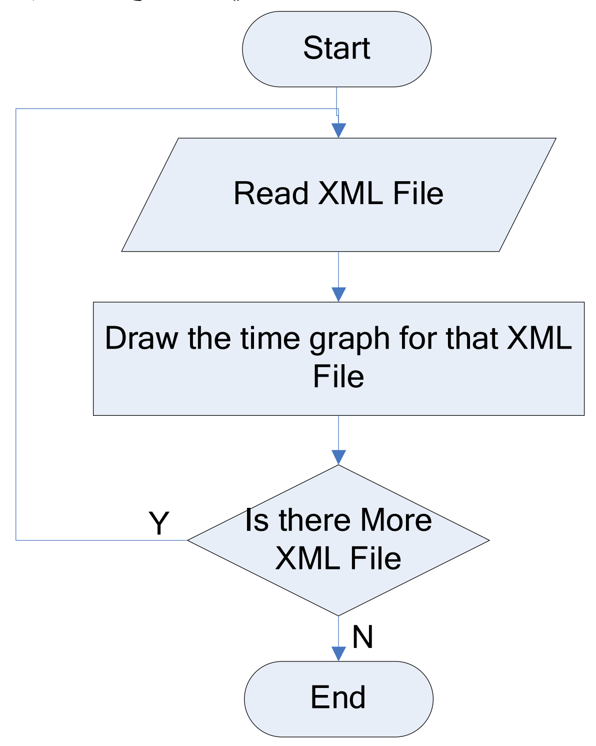
1- اقرأ ملف النتائج المخزن على شكل ملف XML .

2- ارسم المنحني الزمني المعبر عن الملف المقروء .

3- في حال وجود ملف نتائج جديد اذهب إلى الخطوة(2) وإلا اذهب إلى الخطوة(4).

4- النهاية.

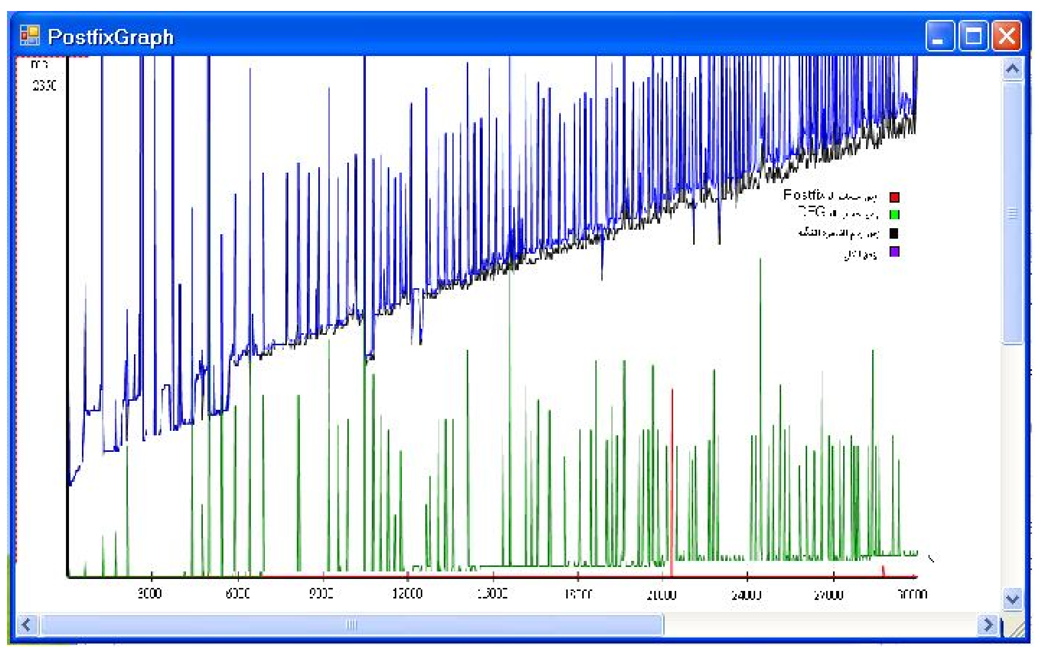
يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح بالشكل(5-7):



الشكل(5-7) المخطط النهجي لعملية مقارنة المنحنيات الزمنية.

ب- الأمر Clear : لمسح المنحنيات الموجودة استعداداً لعملية مقارنة جديدة.

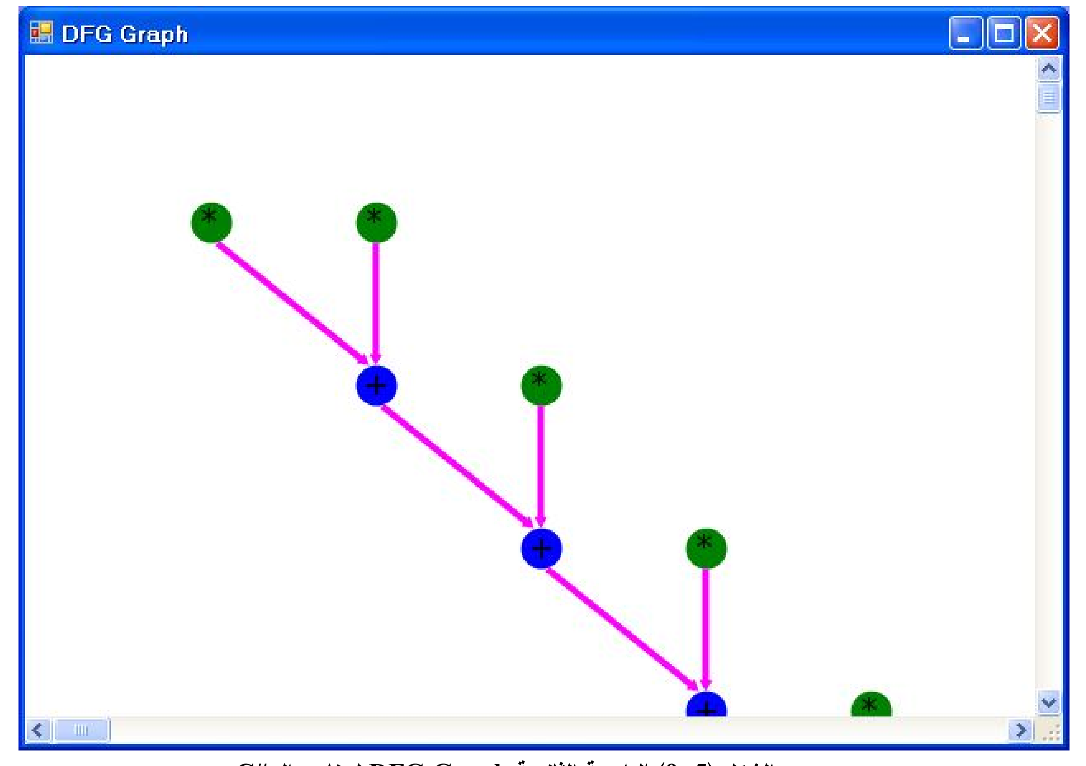
ت- الأمر Generate Time Graph: ينتقل بنا إلى واجهة ثانوية موضحة بالشكل (5-8):



الشكل (5-8) الواجهة الثانوية **PostfixGraph** لبرنامج الـ **C#** .

هذه الواجهة تظهر لنا المنحنيات الزمنية لمرحلتي الترجمة والجدولة للعلاقة بغية معرفة تعقيد الخوارزميات المستخدمة في لغة C# .حيث لم نستخدم عنصر جاهز من مكتبة الـ C# وذلك لإمكانية تطبيق البرنامج على الإصدارات الأحدث من نسخة برنامج الـ C# حيث اعتمدنا على رسم النقاط المعبرة عن المنحني الزمني نقطة نقطة وذلك باحتساب الإحداثيات الدقيقة لها على الشاشة .

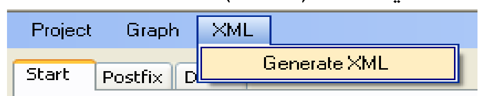
- الأمر DFG Graph : ينتقل بنا إلى الواجهة الثانوية الموضحة بالشكل(5-9):



الشكل (5-9) الواجهة الثانوية **DFG Graph** لبرنامج الـ **C#** .

في هذه الواجهة يتم رسم الشجرة الثنائية المعبرة عن DFG أو SDFG وذلك حسب الاختيار المحدد في الواجهة الرئيسية.

3- القائمة XML : موضحة في الشكل(5-10):



الشكل (5-10) القائمة XML للواجهة الرئيسية لبرنامج الـ C# .

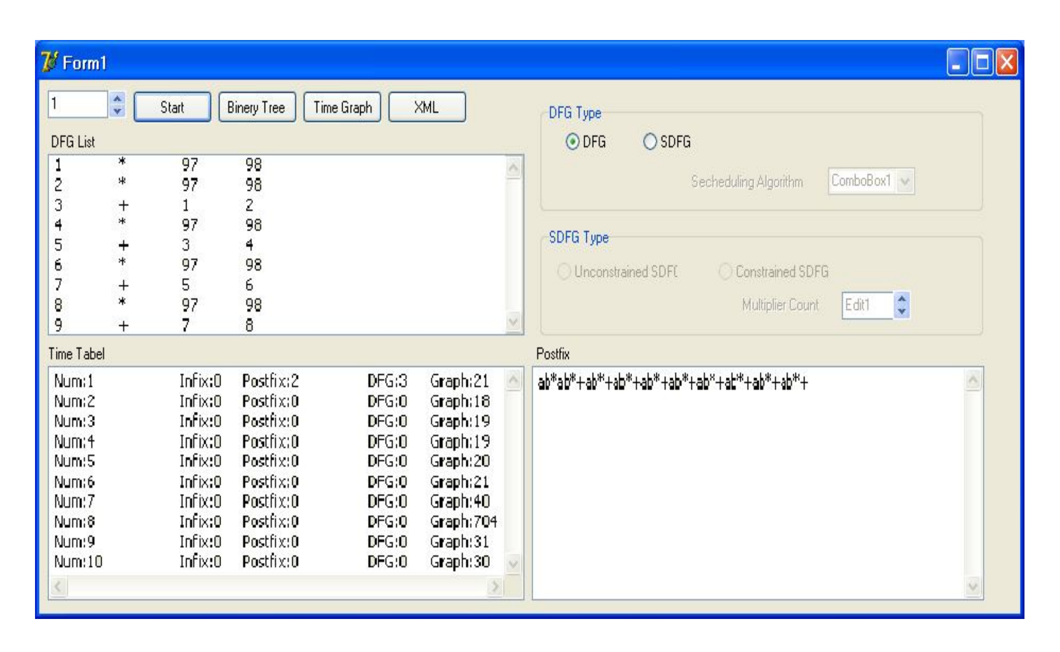
هذه القائمة تحتوي على أمر وحيد وهو:

- الأمر Generate XML : لتخزين النتائج الزمنية التي حصلنا عليها على شكل ملف XML تمهيداً لاستخدامه لاحقاً في عملية المقارنة .

5-2- واجهات البرنامج المكتوب بلغة Delphi :

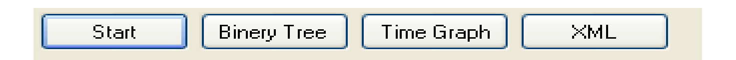
نستعرض فيما يلي واجهة البرنامج المكتوب بلغة الـ Delphi مع شرح مفصل لكل مكوناتها:

يبين الشكل (5-11) الواجهة الرئيسية لبرنامج الـ Delphi :



الشكل (5-11) الواجهة الرئيسية لبرنامج الـ Delphi .

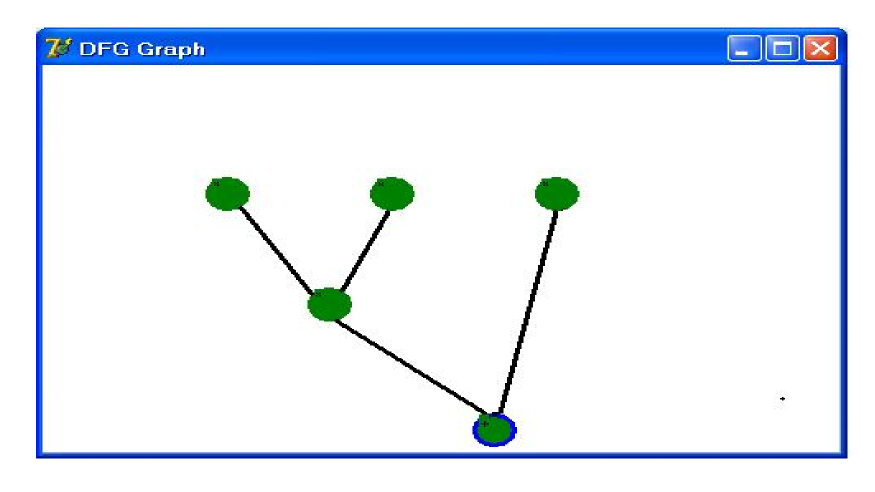
هذه الواجهة تحتوي على أربعة أزرار تقوم بالمهام التالية :



1- الزر Start : هذا الزر يبدأ مرحلة الترجمة أو مرحلة الجدولة حسب الاختيار ، في البداية نحدد القيمة النهائية للعلاقة ومن ثم نختار مرحلة الترجمة أو الجدولة ، في حال اختيار مرحلة

الجدولـة نحـدد بعدها نـوع خوارزميـة الجدولـة المسـتخدمة وفيمـا إذا كانـت بقيـود أم لا مـع تحديـد عـدد الضـوارب ، تظهـر لنـا النتـائج الزمنيـة للتنفيـذ فـي هـذه الواجهـة بالإضـافة إلـى اللائحـة المترابطة وأيضاً صيغة Postfix للعلاقة بقيمتها النهائية.

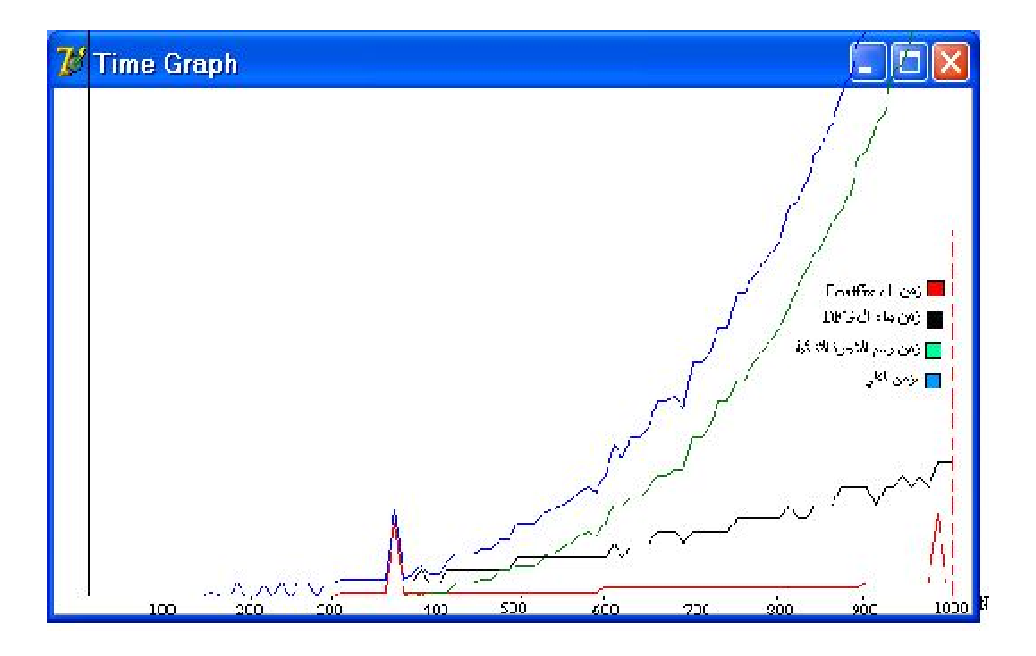
2- الزر Binary Tree : ينتقل بنا إلى واجهة ثانوية موضحة في الشكل (5-12):



الشكل (5-12) الواجهة الثانوية DFG Graph لبرنامج الـ Delphi .

تظهر لنا هذه الواجهة الشجرة الثنائية المعبرة عن اللائحة المترابطة ، كما هو الحال في لغة الـ C# تم استخدام خوارزميات خاصة من أجل رسم الشجرة الثنائية وبالطبع فإن عملية الرسم في لغة Delphi معقدة وأكثر صعوبة من لغة C# بسبب افتقار لغة Delphi إلى المكتبات الرسومية مثل الدوائر.

3- الزر Time Graph : ينتقل إلى واجهة ثانوية موضحة في الشكل (5-13):



الشكل (5-13) الواجهة الثانوية Time Graph لبرنامج الـ Delphi .

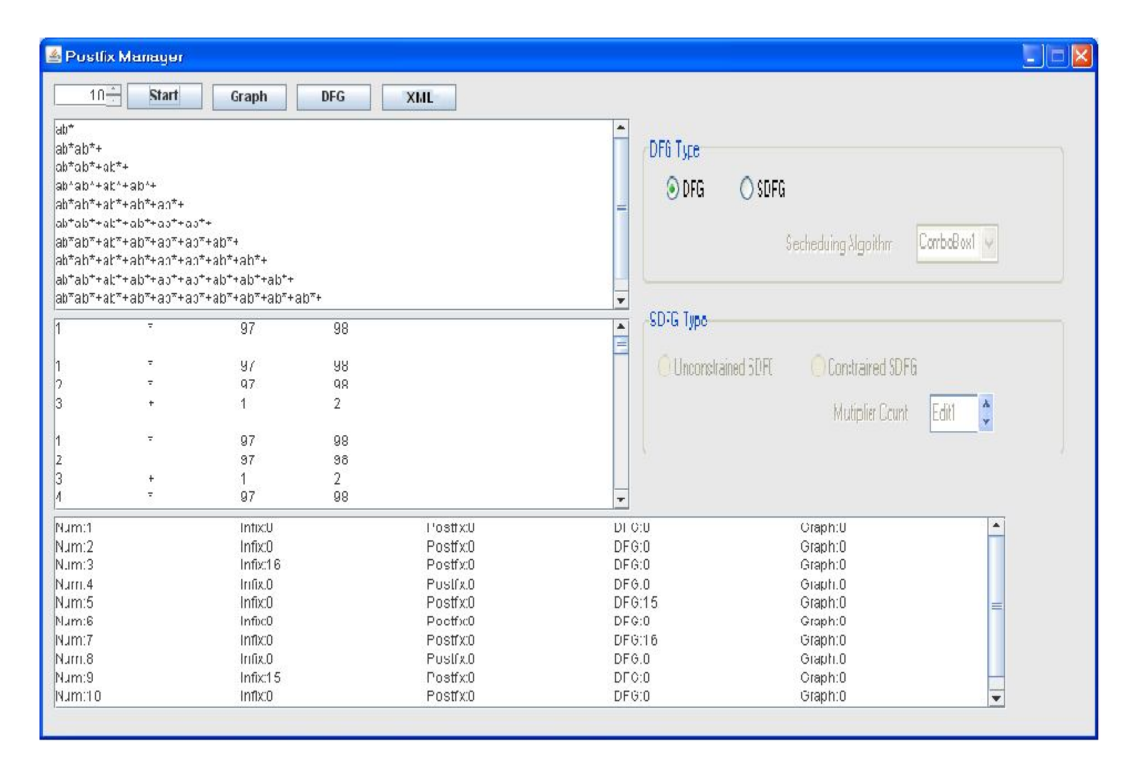
تظهر لنا هذه الواجهة المنحنيات الزمنية للنتائج التي توصلنا لها بغية معرفة درجة تعقيد الخوارزميات في لغة Delphi .

4- الزر XML : : لتخزين النتائج الزمنية التي حصلنا عليها على شكل ملف XML تمهيداً لاستخدامه لاحقاً في عملية المقارنة .

5-3- واجهات البرنامج المكتوب بلغة Java :

نستعرض فيما يلي واجهة البرنامج المكتوب بلغة الـ Java مع شرح مفصل لكل مكوناتها:

يبين الشكل (5-14) الواجهة الرئيسية لبرنامج الـ Java :



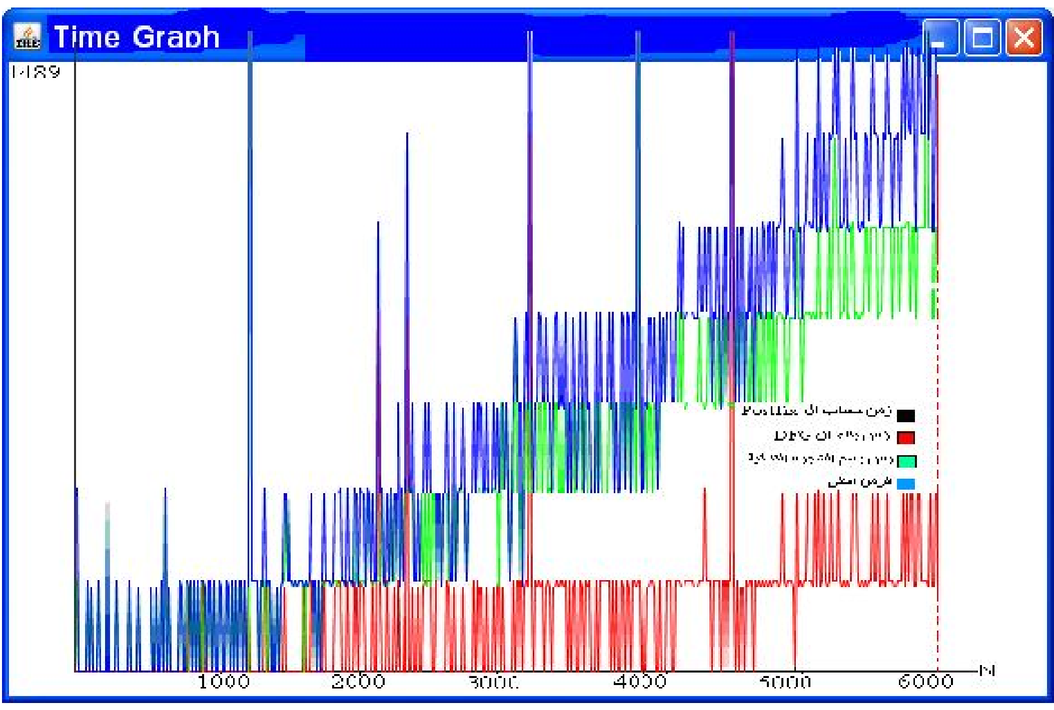
الشكل (5-14) الواجهة الرئيسية لبرنامج الـ Java .

هذه الواجهة تحتوي على أربعة أزرار تقوم بالمهام التالية :



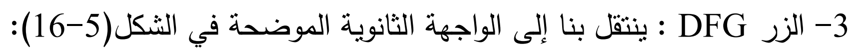
1- الزر Start : هذا الزر يبدأ مرحلة الترجمة أو مرحلة الجدولة حسب الاختيار ، في البداية نحدد القيمة النهائية للعلاقة ومن ثم نختار مرحلة الترجمة أو الجدولة ، في حال اختيار مرحلة الجدولة نحدد بعدها نوع خوارزمية الجدولة المستخدمة وفيما إذا كانت بقيود أم لا مع تحديد عدد الضوارب ، تظهر لنا النتائج الزمنية للتنفيذ في هذه الواجهة بالإضافة إلى اللائحة المترابطة وأيضاً صيغة Postfix للعلاقة بقيمتها النهائية.

2- الزر Graph : ينتقل بنا إلى واجهة ثانوية موضحة في الشكل (5-15):

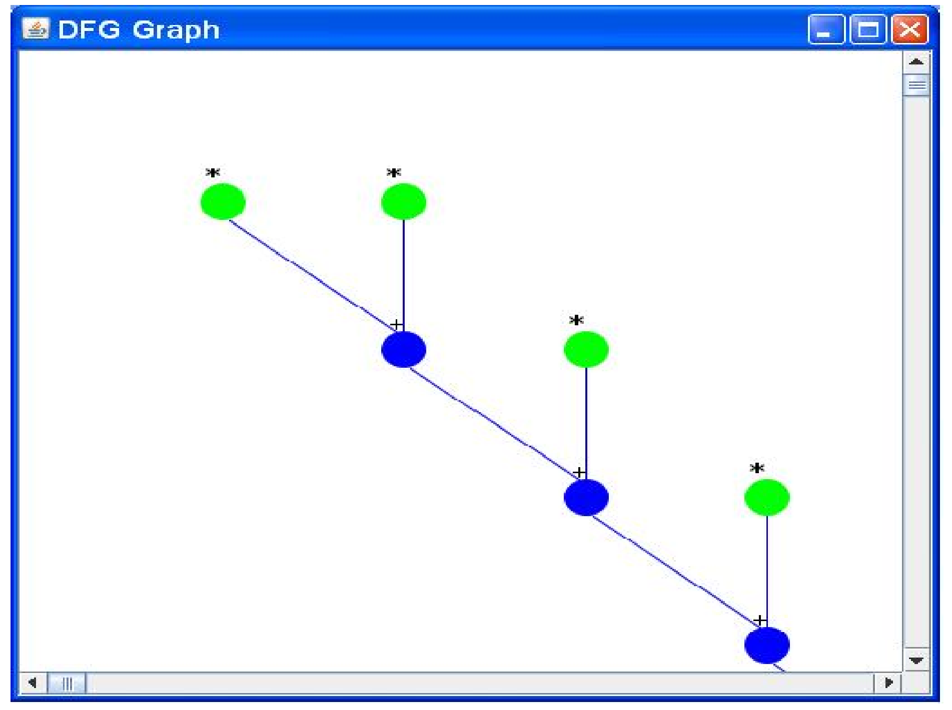


الشكل (5-15) الواجهة الثانوية Time Graph لبرنامج الـ Java .

تظهر لنا هذه الواجهة المنحنيات الزمنية للنتائج التي توصلنا لها بغية معرفة درجة تعقيد الخوارزميات في لغة Java .



3- الزر DFG : ينتقل بنا إلى الواجهة الثانوية الموضحة في الشكل(5-16):



الشكل (5-16) الواجهة الثانوية DFG Graph لبرنامج الـ Java .

تظهر لنا هذه الواجهة الشجرة الثنائية المعبرة عن اللائحة المترابطة ، وكما هو الحال في لغة الـ C# تم استخدام خوارزميات خاصة من أجل رسم الشجرة الثنائية وبالطبع فإن عملية الرسم في لغة Java أبسط وأكثر سهولة من لغة Delphi بسبب وفرة هذه اللغة بالمكتبات الداعمة للرسوميات مثل لغة C#.

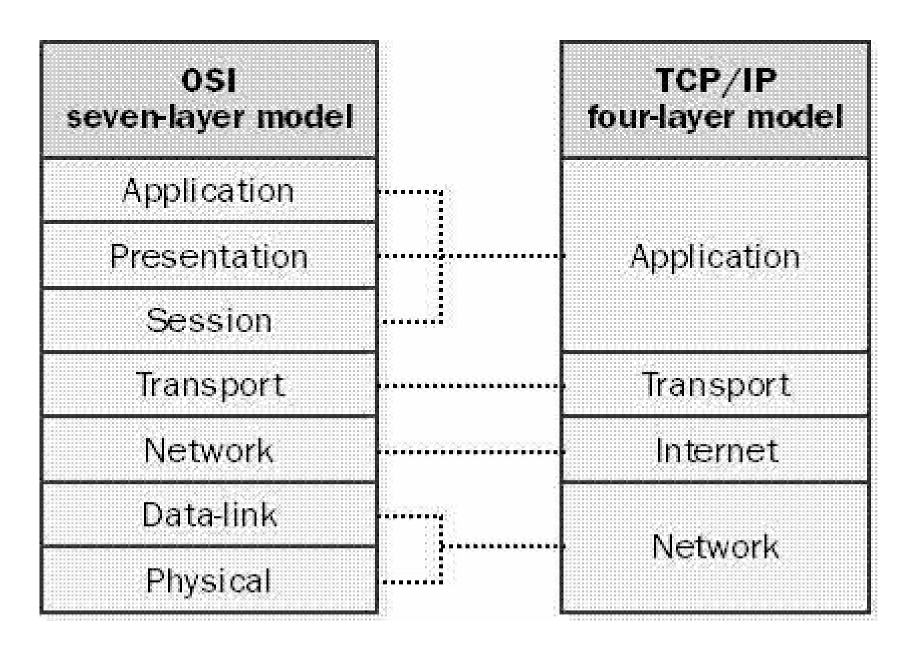
4- الزر XML : لتخزين النتائج الزمنية التي حصلنا عليها على شكل ملف XML تمهيداً لاستخدامه لاحقاً في عملية المقارنة .

الفصل السادس

مرحلة المعالجة الموزعة

6-1- مقدمة في برمجة الشبكات والبروتوكول TCP/IP:[26,27,28]

من المعروف أن الشبكة هي مجموعة من الأجهزة المتصلة مع بعضها عبر وسيلة اتصال معينة ومن هنا سيندرج لدينا التقسيم المعروف لمنظمة OSI لعملية الاتصال والتي تمر بسبع طبقات لكل طبقة منها وظيفة معينة وتم اختصارها إلى أربعة طبقات (خمسة في بعض الكتب) في بروتوكول TCP/IP ، ويبين الشكل(6-1) هذه الطبقات:



الشكل(6-1) طبقات البروتوكول TCP/IP.

يلزم ما يلي في الجهاز المرسل Client لإجراء عملية الاتصال بين Client و Server:

تبدأ عملية توليف الرسالة المرسلة في الـ Application Layer ووظيفتها هنا التعامل مع الرسالة نفسها وتحويلها من صيغة نصية إلى Data يمكن إرسالها عبر الشبكة، ففي برمجيات الدردشة Chat يتم تحويل النص المكتوب إلى ASCII Code ثم إلى مجموعة من Binary Code توضع في مصفوفة لتجهيزها وإرسالها عبر Socket والذي يربط طبقة Application Layer مع بقية طبقات TCP/IP ، فيما يلي نبين طريقة تحويل الرسالة المكتوبة كنص باستخدام الـ ASCIIencoding Class إلى Byte Array وذلك بلغة C# :[29]

```
String str=Console.ReadLine();
ASCIIEncoding asen= new ASCIIEncoding();
byte[] ba=asen.GetBytes(str);
```

في نموذج OSI تم تقسيم Upper Layer إلى ثلاث طبقات:

- Application للتعامل مع البرنامج نفسه أو ما يسمى User Interface .

- Presentation لتحويل البيانات المرسلة إلى ASCII .

- Session وفيها يتم البدء بعملية التخاطب بين الجهازين والتعريف ببعضهما البعض (فتح الجلسة).

أما بروتوكول TCP/IP فاكتفى بوجود طبقة Application والتي تقوم بعمل الطبقات الثلاث الأولى في OSI ، في Session Layer يتم التعرف وفتح الجلسة بعدة خطوات وهي كما يلي:

1- إجراء الاتصال المبدئي بجهاز Server عبر IP والـ Port المحدد وذلك بعد تحديد عملية الاتصال سواء عبر UDP أو TCP .

2- التعريف بنفسه أي بالجهاز المرسل.

3- قبول أو رفض الجلسة ويتم ذلك بإرسال الموافقة على فتح الجلسة أو رفضها.

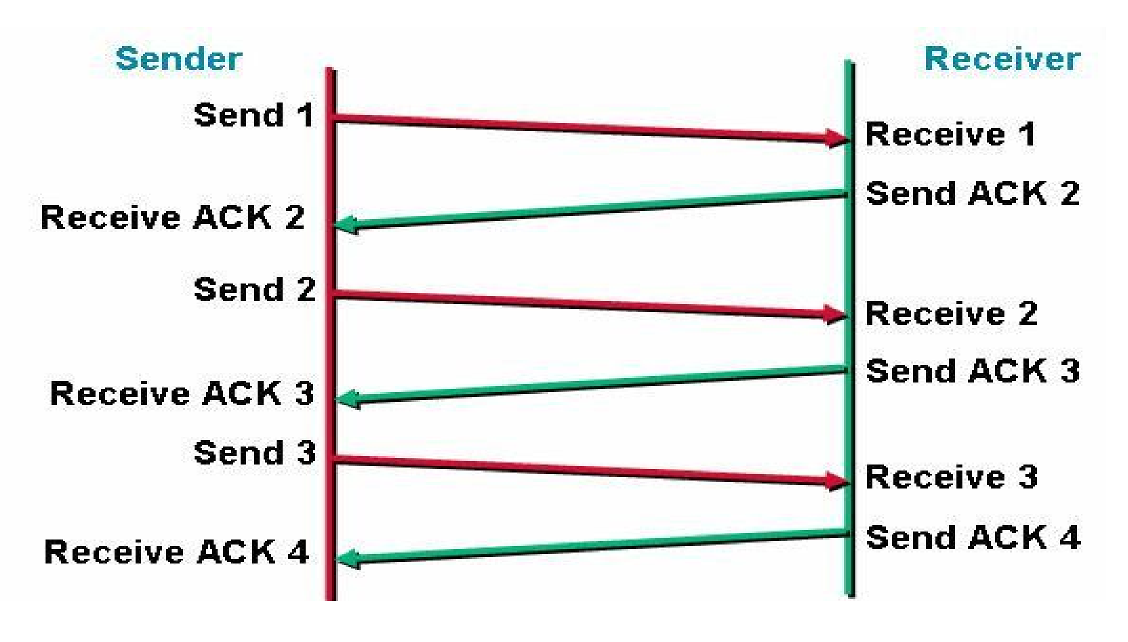
4- بدأ الجلسة وقيام الـ Server بعمل Listening على الـ Port الخاص بالبرنامج.

عندما يتم الموافقة على فتح الجلسة والبدء بعملية التخاطب يقوم الجهاز المرسل Client بتحميل الرسالة إلى الطبقة الأخرى وهي هنا طبقة Transport وفي هذه الطبقة يتم تحديد طبيعة الاتصال TCP– Connection Protocol أو عبر UDP – Connectionless Protocol ، ففي البروتوكول الأول يتم تحديد طرفين وهما المرسل والمستقبل و Port الاتصال، أما في البروتوكول الثاني UDP يتم تحديد الطرف المرسل والمستقبل (اختياري) أي أنه يمكن عمل Broadcast بدون تحديد جهة معينة لاستقبال الرسالة أي أن أي شخص يقوم بالتنصت عبر هذا الـ Listening Port يستطيع استقبال الرسالة، وهنا مثال يوضح عمل هذه الطبقة باستخدام الـ TCP Protocol وذلك بلغة C# :

C#:

```
TcpClient tcpclnt = new TcpClient();
tcpclnt.Connect("192.168.0.2",8001);
```

وتتم عملية التحقق من الوصول في الـ TCP كما هو موضح في الشكل(6-2):



الشكل(6-2) عملية التحقق من الوصول في بروتوكول TCP.

إذ أنه في كل عملية إرسال يتم إرسال رد Acknowledgment إلى المرسل يخبره فيها بوصول الرسالة، ويرسل في الـ Header Acknowledgment رقم الـ Packet الذي تم استقباله بنجاح ويسمى Ack ID .

ولإرسال الرسالة عبر الشبكة نستخدم في الـ Net. صنف جاهز Class يقوم بهذه العملية ويسمى NetworkStream وهو المسؤول عن التعامل مع وسيلة الاتصال وإرسال الرسالة إلى الطرف المعني على شكل Stream Data ، أو باستخدام الـ Socket نفسه وكمثال على ذلك في لغة C# :

```
NetworkStream mynetsream = tcpclnt.GetStream ();
StreamWriter myswrite = new StreamWriter (mynetsream);
myswrite.WriteLine("Your Message");
```

وبعد ذلك تسلم إلى Network Layer إذ تتم عنونة الرسالة ووضع عنوان المرسل والمستقبل عليها وتسلم إلى الطبقة الأدنى ليتم إرسالها عبر Physical Tunnel .

أما بالنسبة للجهاز المستقبل الـ Server فيقوم بالمرور على نفس الطبقات ولكن بالعكس حيث يستلم كرت الشبكة الـ Bits لتحول إلى Data Link ثم Network ثم Transport ثم Application وفيها تحول من Binary إلى ASCII ومن ASCII إلى Text ، وهذا الكود يوضح مبدأ عمل الـ Server في لغة #C :

```
TcpListener myList=new TcpListener("127.0.0.1",8001);
myList.Start();
Socket s=myList.AcceptSocket();
byte[] b=new byte[100];
int k=s.Receive(b);
for (int i=0;i<k;i++)
Console.Write(Convert.ToChar(b[i]));
s.Close()
```

6-2- Connectionless Sockets Via UDP :[30,31]

بينا في الفقرة السابقة أن برتوكول TCP هو بروتوكول موجه وهذا يعني أنه يلزم احتواء الـ Header الخاص به على عنوان المرسل وعلى عنوان المستقبل كما يلزم أيضاً القيام بعمليات التحقق Authentication ويدعم عمليات التحقق من الوصول والتسليم بشكل صحيح (هذه الشروط مهمة عندما نريد تطبيق مرحلة التوزيع على موضوع الدراسة)، أما في حال كانت هذه الشروط غير مهمة إي إذا أردنا أن نقوم بعملية بث إذاعي Broadcast للرسالة دون الاهتمام بمن سوف يستلم الرسالة وما يهمنا فقط هو سرعة الإرسال والاستقبال عندها نتوجه إلى بروتوكول UDP User Datagram Protocol ويسمى أيضاً Connectionless Protocol .

في هذا البروتوكول نستطيع عمل ما يسمى Broadcast و Multicast (Broad يعني الإرسال إلى الكل أما Multi فيعني الإرسال إلى المجموعة) يوجد شرط وحيد يلزم أن نأخذه بعين الاعتبار عند استخدام الـ UDP لعملية البث Broadcast وهو أن الشبكة التي نريد عمل بث لها تتصل معها بشكل مباشر Direct Connection أي بدون وجود Router بين المرسل والمستقبل لأن الـ Router يمنع عمليات البث الإذاعي Broadcast حيث يلزم أن تكون الشبكة ضمن الـ Range

Class سواءً A أو B أو C ، ومن المعروف أيضاً أن عنوان الـ IP مقسم إلى جزأين الأول مخصص لشبكة Network والثاني مخصص لـ Host كما هو موضح في الشكل (3-6):

	8 bits	8 bits	8 bits	8 bits
Class A:	Network	Host	Host	Host
Class B:	Network	Network	Host	Host
Class C:	Network	Network	Network	Host
Class D:	Multicast start 224.0.0.1			
Broadcast:	Network	255 11111111	255 11111111	255 11111111

الشكل(3-6) عنوان IP في بروتوكول UDP.

لاستخدام الـ UDP في الـ Net. يلزم أولاً تعريف Space Name System.Net والـ Socket System.Net ، في الـ TCP يلزم تعريف رقم الـ Port والعنوان للجهاز المستقبل أما في حالة UDP نستطيع تعريفه كما هو في TCP ونستطيع عمل Broadcast باستخدام IP Address.Any بعد اشتقاق كائن من الصنف IPEndPoint (وتعني نقطة الهدف) ونستطيع أيضاً عدم تحديد رقم الـ Port باستخدام الـ Bind Method حيث يتم تعريفها بـ 0 .

في المثال التالي يتم فتح الـ Port 5020 والتنصت عليها ثم استلام الرسالة عبر هذا الـ Port المخصص لعملية البث الإذاعي ذو الرقم 0 :

C#:

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5020);
```

لتحديد نوع البروتوكول المستخدم يتم ذلك كما يلي:

C#:

```
Socket newsock = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp)
```

ثم تمرير نقطة الهدف ورقم الـ Port إلى Send Method .

الـ Bind Method يتم وضعها في الطرف المستقبل فقط إذ تربط الـ IP Address ورقم الـ Port بالـ Socket :

C#:

```
newsock.Bind(ipep);
```

الآن تم استقبال الرسالة ونريد بثها إلى كل من يتصل مع الـ Server على الـ Port السابقة ولعمل ذلك يلزم أولاً تعريف نقطة الهدف كما يلي:

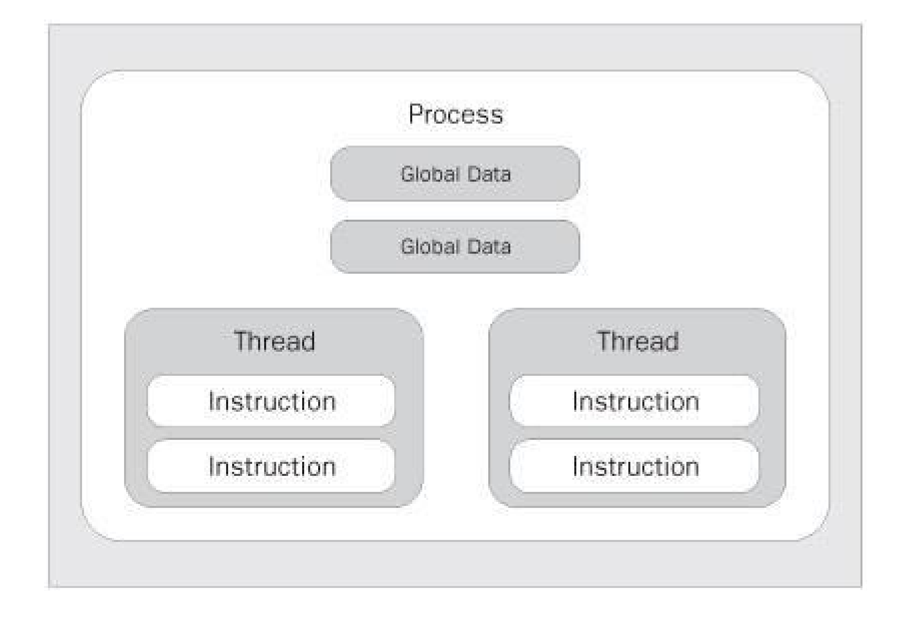
C#:

```
IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
EndPoint Remote = (EndPoint)(sender);
```

الآن الجزء الخاص بالـ Client ، يقتصر العمل هنا على قيام الـ Client بإنشاء جلسة مع الـ Server وذلك بعد تعريفه بالـ IPEndPoint ورقم الـ Port وكما تم في السابق إلا أن الاختلاف هو في الوظيفة إذ يقتصر فقط على استقبال الرسالة من الـ Server وإرسال أي رسالة له عبر الـ Port المخصص لهذه العملية.

6-3- نظرة عامة على Threading :[26,29]

قدمت لنا الـ Net. خاصية مهمة وهي استخدام خاصية الـ Threading التي تسمح بالمعالجة المتوازية على نفس المعالج وذلك من خلال تقسيم المهام على المعالج وعمل Session منفصلة لكل برنامج وهو ما يسمى بالـ Multitasking ، وهنا لا يؤثر البرنامج على موارد النظام بشكل كبير كما أن الـ Loop ستعمل في Thread منفصل عن الـ Thread الخاص بالـ Form كما هو موضح في الشكل (6-4):



الشكل (6-4) يبين خاصية الـ Thread في الـ Net. .

نلاحظ أنه قبل إضافة الـ Thread كان الـ Loop يعمل على منطقة الـ Global Area وهذا هو سبب البطء الشديد، وبعد استخدام الـ Thread تم عمل Session خاص للـ Loop بحيث يعمل بشكل متوازي مع البرنامج.

إن ميزة الـ Thread رائعة جداً إذ تمكننا من تشغيل أكثر من Thread وفي نفس الوقت وفي نفس البرنامج وهو ما يسمى بالـ Multithreading .

6-4- خوارزميات مرحلة المعالجة الموزعة:[28,31]

في هذه المرحلة سيتم توزيع مرحلة الترجمة على مجموعة من الأجهزة من أجل مقارنة النتائج مع نتيجة برنامج الـ C# المطبق على حاسب واحد، وقد استخدمنا بروتوكول TCP/IP وذلك لكونه بروتوكول مأمون وموثوق بمعنى أننا لا نقوم بإرسال أي أمر آخر من Server إلى Client إلا إذا قام الـ Client بالرد على Server بأنه انهى العمل المطلوب منه، على عكس بروتوكول UDP الذي لايهتم بقيام الـ Client بالرد على الـ Server .

قمنا ببناء برنامجين مكتوبين بلغة الـ C# البرنامج الأول تم تطبيقه على جهاز Server والبرنامج الثاني تم تطبيقه على الأجهزة Clients ، تم الاستفادة من إمكانيات مخبر الـ Open Source في كلية الهندسة الكهربائية حيث تم توصيل جميع أجهزة المخبر عن طريق شبكة جامعة حلب كخطوة أولى ثم توصيل هذه الأجهزة عن طريق شبكة داخلية.

6-4-1- خوارزمية التوزيع المطبقة على الـ Server :

من مهام المخدم تقسيم العلاقة العامة التي من الشكل :

$$y(n) = \sum_{k=1}^{N} a_k * b_k$$

بشكل متساوي بين أجهزة الزبائن كما يقوم بحساب الزمن الكلي للعملية عن طريق حساب أزمنة التخاطب مع الزبائن وحساب زمن المعالجة بالإضافة إلى حساب زمن الإجابة من الزبائن بإتمام المهمة، وذلك باستخدام الخوارزمية التالية:

1- ادخل عدد الزبائن C والقيمة النهائية للعلاقة.

2- اجعل I=1 .

3- احسب طول العلاقة الجديد الذي سيتم إرساله لجميع الزبائن J=I/C .

4- ارسل طول العلاقة J لجميع الزبائن ليقوموا بإيجاد اللائحة المترابطة DFG وابدأ بحساب الزمن.

5- انتظر حتى يقوم جميع الزبائن بالرد بإنهاء المهمة وذلك بإرسال لائحة DFG .

6- اجمع اللوائح المترابطة المرسلة من الزبائن في لائحة مترابطة نهائية Final List .

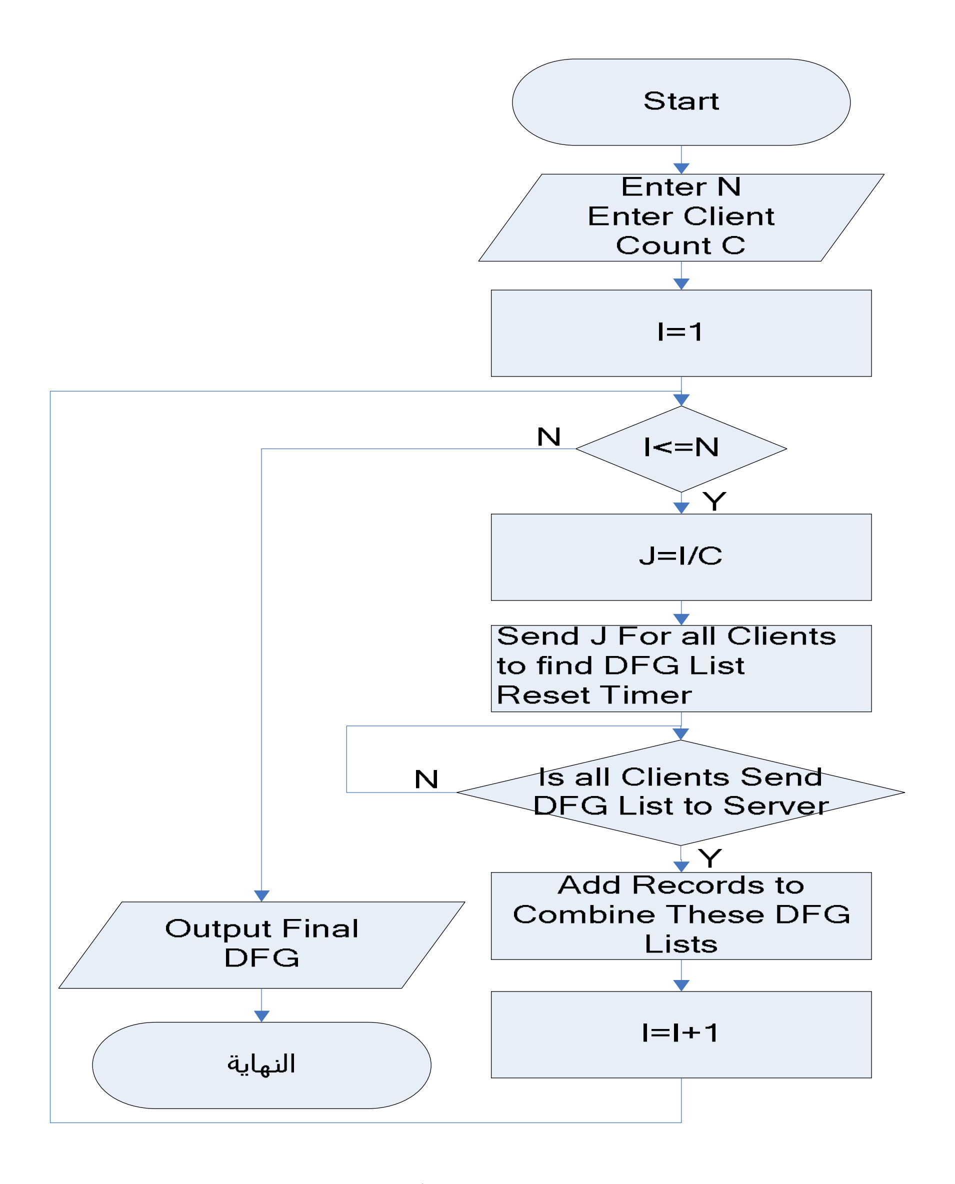
7- أنهي حساب الزمن Ti .

8- I=I+1 .

9- طالما I<=N عندها اذهب إلى الخطوة(3).

10- اطبع اللائحة المترابطة النهائية Final List .

يمكن تلخيص الخوارزمية السابقة بالمخطط النهجي الموضح في الشكل (6-5):



الشكل(6-5) المخطط النهجي لخوارزمية التوزيع في طرف الـ Server .

6-4-2- خوارزمية التوزيع المطبقة على الـ Client :

من ناحية أخرى يقوم الزبون باستلام الأمر من المخدم لحساب العلاقة بالطول الذي يحدده المخدم ومن ثم يقوم بإعلام المخدم بانتهاء المهمة ، إن الخوارزمية التي يعتمدها الزبون موضحة بالخطوات التالية:

1- إذا كان المخدم في حالة تنصت اذهب إلى الخطوة(2).

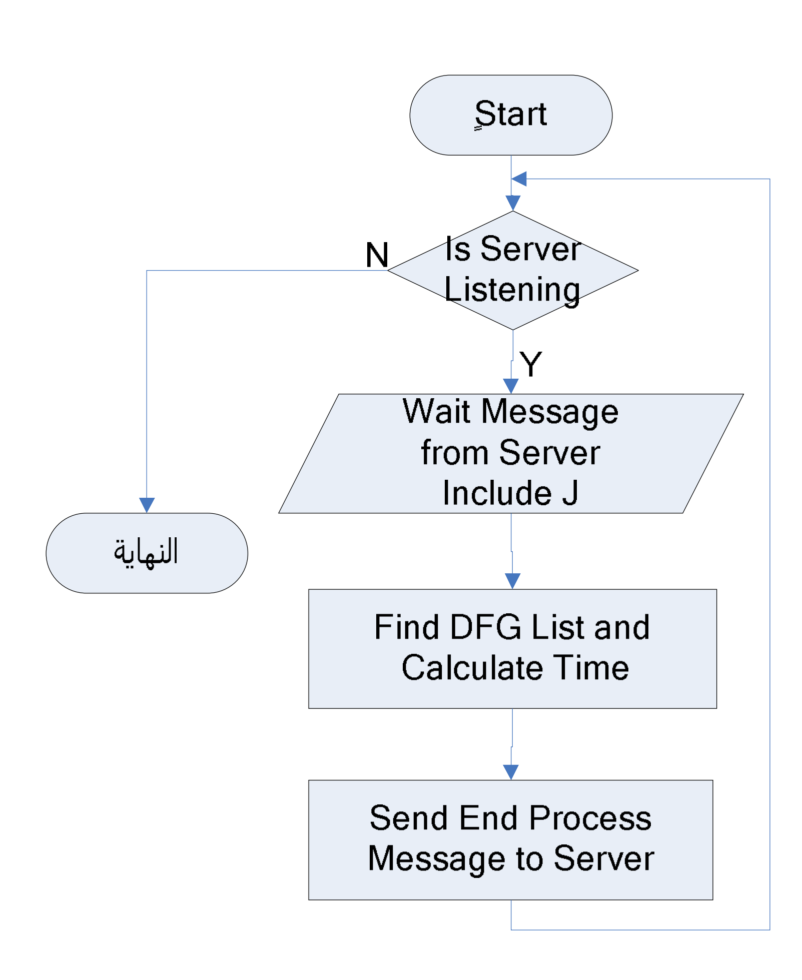
2- أجري عملية اتصال مع المخدم.

3- انتظر رسالة من المخدم تتضمن طول العلاقة J المراد حساب اللائحة المترابطة DFG من أجلها.

4- أوجد اللائحة المترابطة DFG واحسب الزمن الكلي.

5-ارسل للمخدم رسالة تفيد بإنتهاء العملية ثم اذهب إلى الخطوة(3).

يمكن تلخيص الخوارزمية بالمخطط النهجي الموضح بالشكل (6-6):



الشكل(6-6) المخطط النهجي لخوارزمية التوزيع في طرف الـ Client .

الفصل السابع

تأثير ونتائج المعالجة الموزعة على مرحلة الترجمة

7-1- تأثير المعالجة الموزعة على مرحلة الترجمة :

سيتم التطبيق من أجل العلاقة العامة التالية المشتقة من الصيغة العامة للمرشح الرقمي IIR :

$$y(n) = \sum_{k=1}^{N} a_k * b_k$$

هذا الفصل يتعرض لمقارنة عدة مراحل مختلفة من التوزيع من خلال منحنيات بيانية تبين زمن المعالجة وتغيرها بتغير عدد الأجهزة المستخدمة ، بالإضافة إلى تأثير المعالجة الموزعة على شكل الشجرة الثنائية الممثلة للنظام وذلك لبرنامج الـ C#.

في البداية سنوضح أثر التوزيع على شكل الشجرة الثنائية الممثلة للنظام من أجل N=10 حيث تصبح العلاقة العامة من الشكل التالي:

$$Y=a_1*b_1+a_2*b_2+a_3*b_3+..........+a_{10}*b_{10}$$

7-1-1- الشجرة الثنائية الممثلة للنظام عند استخدام حاسب واحد:

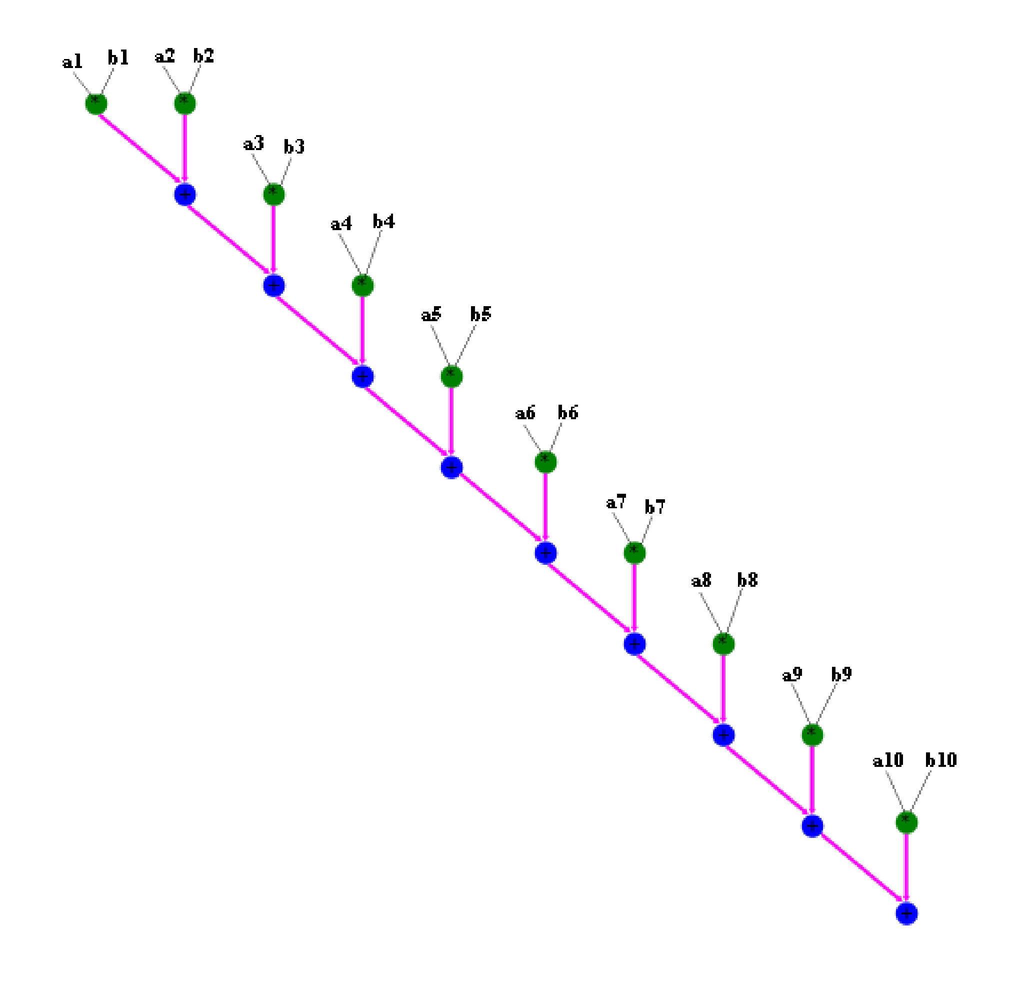
بينا في الفصل الثاني كيفية ترجمة العلاقة على حاسب واحد لنحصل على اللائحة المترابطة DFG ،من أجل N=10 كانت اللائحة المترابطة DFG كما هو موضح في الجدول (7-1):

الجدول (7-1) اللائحة المترابطة المعبرة عن DFG بالنسبة لحاسب واحد.

Rec Num	Var left	Next left	opration	Next right	Var right
1	a1	Null	*	Null	b1
2	a2	Null	*	Null	b2
3		1	+	2	
4	a3	Null	*	Null	b3
5		3	+	4	
6	a4	Null	*	Null	b4
7		5	+	6	
8	a5	Null	*	Null	b5
9		7	+	8	

10	a6	Null	*	Null	b6
11		9	+	10	
12	a7	Null	*	Null	b7
13		11	+	12	
14	a8	Null	*	Null	b8
15		13	+	14	
16	a9	Null	*	Null	b9
17		15	+	16	
18	a10	Null	*	Null	b10
19		17	+	18	

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية التالية الموضحة بالشكل (7−1) وذلك من أجل حاسب واحد:



الشكل (7−1) الشجرة الثنائية DFG من أجل حاسب واحد

7-1-2- الشجرة الثنائية الممثلة للنظام عند استخدام مخدم وزبونين:

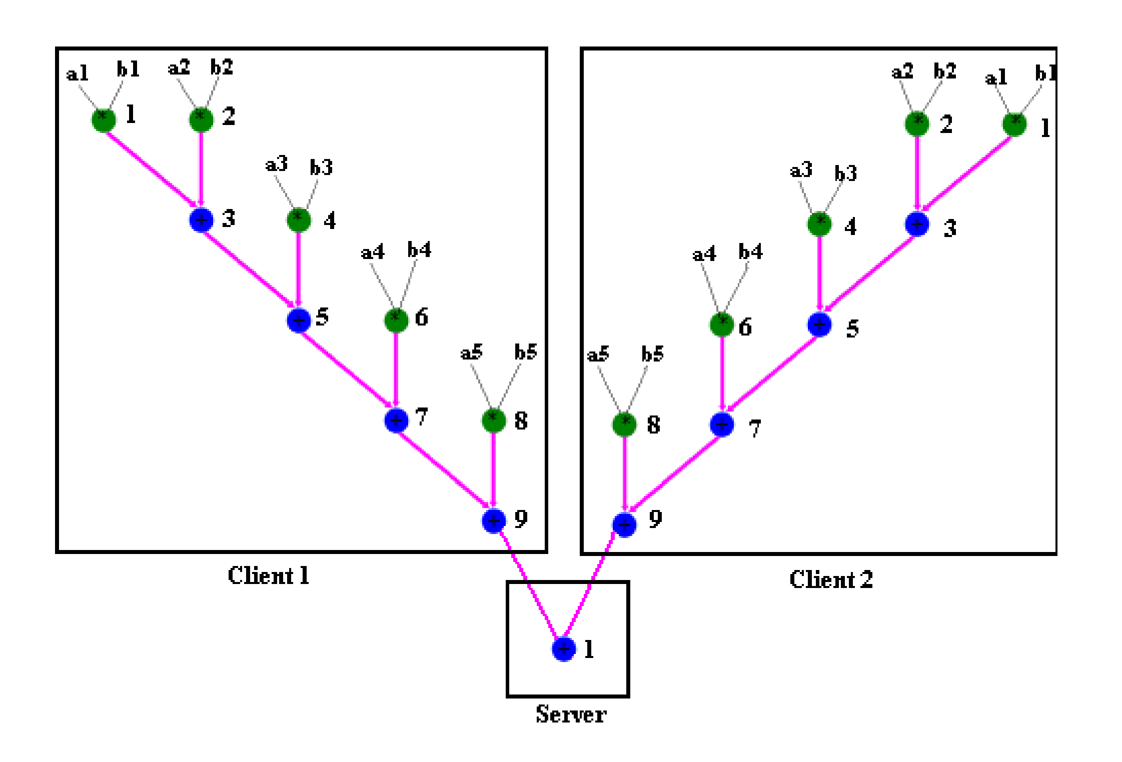
المرحلة التالية هي في قيام المخدم بتقسيم العلاقة السابقة ذات الطول N على زبونين حيث يقوم المخدم بحساب طول العلاقة الجديد J=N/2=5 لإرساله إلى الزبونين ليقوما بترجمة العلاقة بطولها الجديد J كما وضحنا سابقاً في الفصل الثاني .

عندما يقوم الزبونين بإعلام المخدم بإنتهاء عملية الترجمة ويرجعا له رقم آخر سجل في اللائحة المترابطة DFG ، يقوم المخدم بإنشاء اللائحة المترابطة النهائية Final List التي تقوم بدمج اللوائح المترابطة المرسلة من الزبائن، كما هو موضح في الجدول (7-2):

الجدول(7-2) اللائحة المترابطة النهائية Final List في طرف المخدم.

Rec Num	Next left	opration	Next right
1	Rec9(C-1)	+	Rec9(C-2)

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية التالية الموضحة بالشكل (7-2) وذلك من أجل زبونين ومخدم :



الشكل (7-2) الشجرة الثنائية DFG من أجل زبونين ومخدم.

7-1-3- الشجرة الثنائية الممثلة للنظام عند استخدام مخدم وثلاثة زبائن:

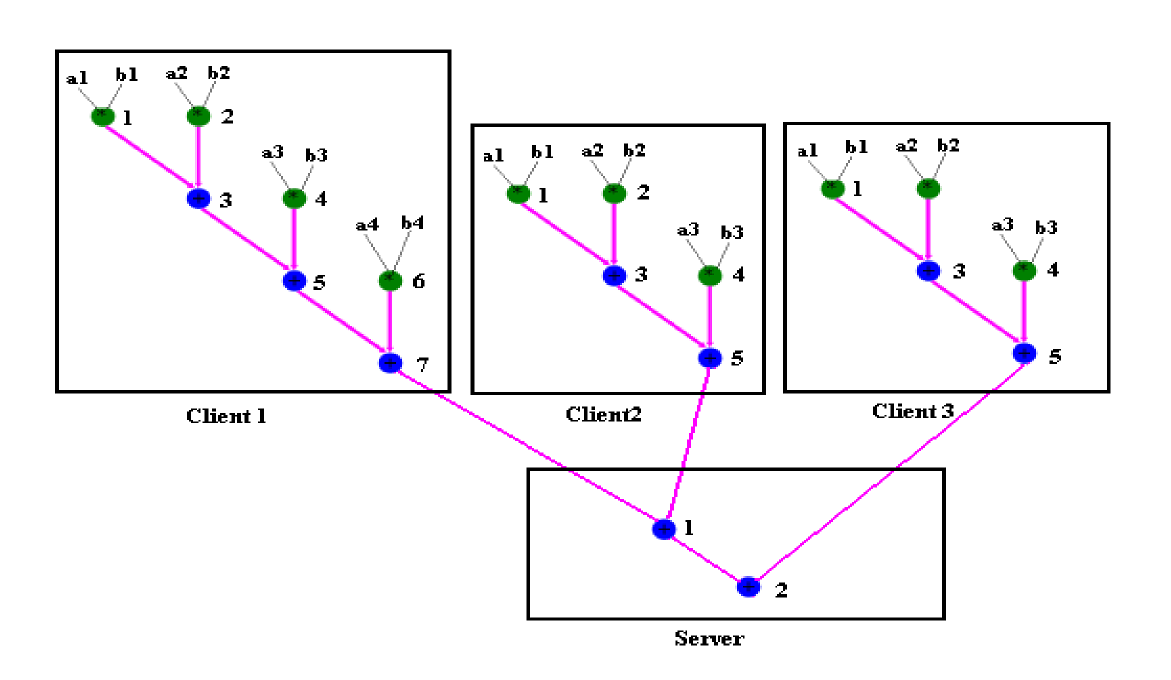
المرحلة التالية هي في قيام المخدم بتقسيم العلاقة السابقة ذات الطول N على ثلاثة زبائن حيث يقوم المخدم بحساب طول العلاقة الجديد الذي سيتم إرساله إلى الزبائن ليقوموا بترجمة العلاقة بطولها الجديد J .

تقوم الزبائن بإعلام المخدم بإنتهاء عملية الترجمة وإرجاع رقم آخر سجل له في اللائحة المترابطة DFG ، يقوم المخدم بإنشاء اللائحة المترابطة النهائية Final List التي تقوم بدمج اللوائح المترابطة المرسلة من الزبائن، كما هو موضح في الجدول (7-3):

الجدول(7-3) اللائحة المترابطة النهائية Final List في طرف المخدم.

Rec Num	Next left	opration	Next right
1	Rec7(C-1)	+	Rec5(C-2)
2	1	+	Rec5(C-3)

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية التالية الموضحة بالشكل (7-3) وذلك من أجل ثلاثة زبائن ومخدم :



الشكل (7-3) الشجرة الثنائية DFG من أجل ثلاثة زبائن ومخدم.

7-1-4- الشجرة الثنائية الممثلة للنظام عند استخدام مخدم وأربعة زبائن:

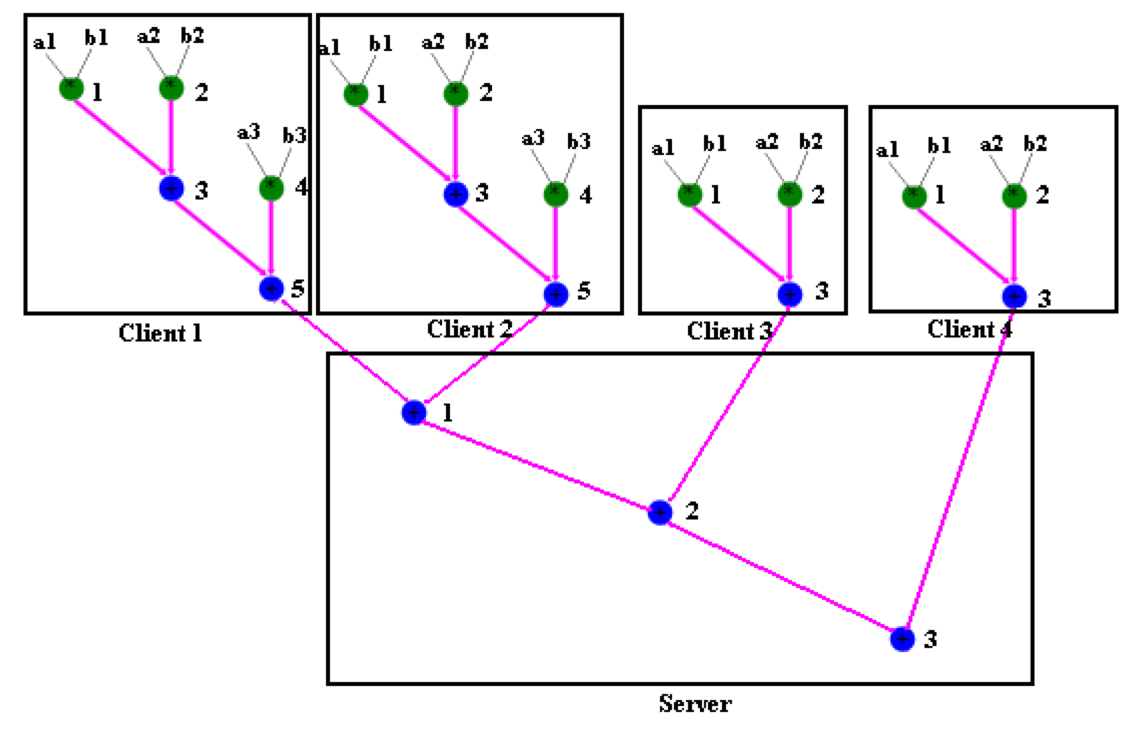
المرحلة التالية هي في قيام المخدم بتقسيم العلاقة السابقة ذات الطول N على أربعة زبائن حيث يقوم المخدم بحساب طول العلاقة الجديد الذي سيتم إرساله إلى الزبائن ليقوموا بترجمة العلاقة بطولها الجديد J .

عندما يقوم الزبائن بإعلام المخدم بإنتهاء عملية الترجمة وإرجاع رقم آخر سجل له في اللائحة المترابطة DFG ، يقوم المخدم بإنشاء اللائحة المترابطة النهائية Final List التي تقوم بدمج اللوائح المترابطة المرسلة من الزبائن، كما هو موضح في الجدول (7-4):

الجدول(7-4) اللائحة المترابطة النهائية Final List في طرف المخدم.

Rec Num	Next left	opration	Next right
1	Rec5(C-1)	+	Rec5(C-2)
2	1	+	Rec3(C-3)
3	2	+	Rec3(C-4)

من اللائحة المترابطة السابقة نحصل على الشجرة الثنائية التالية الموضحة بالشكل (7-4) وذلك من أجل أربعة زبائن ومخدم :



الشكل (7-4) الشجرة الثنائية DFG من أجل أربعة زبائن ومخدم.

7-2- نتائج المعالجة الموزعة على شبكة جامعة حلب:

تم توزيع العمل السابق (مرحلة الترجمة) على مجموعة من الأجهزة من أجل مقارنة النتائج مع نتيجة برنامج الـ C# المطبق على حاسب واحد.

استخدمنا مخدم بالمواصفات التالية:

Intel® Celeron® D CPU 5.00 GHz,160 GByte HDD,1.0 GByte RAM.

أما الزبائن فهي بالمواصفات التالية:

Intel® Celeron® D CPU 3.46GHz,60 GByte HDD,504 MByte RAM.

قمنا بالمراحل التالية من توزيع العمل:

1- مقارنة نتائج توزيع العمل على حاسبين من أجل N=1..500000 وذلك بخطوة مقدارها 1 في أول مرة ومن ثم خطوة مقدارها 10 في ثاني مرة .

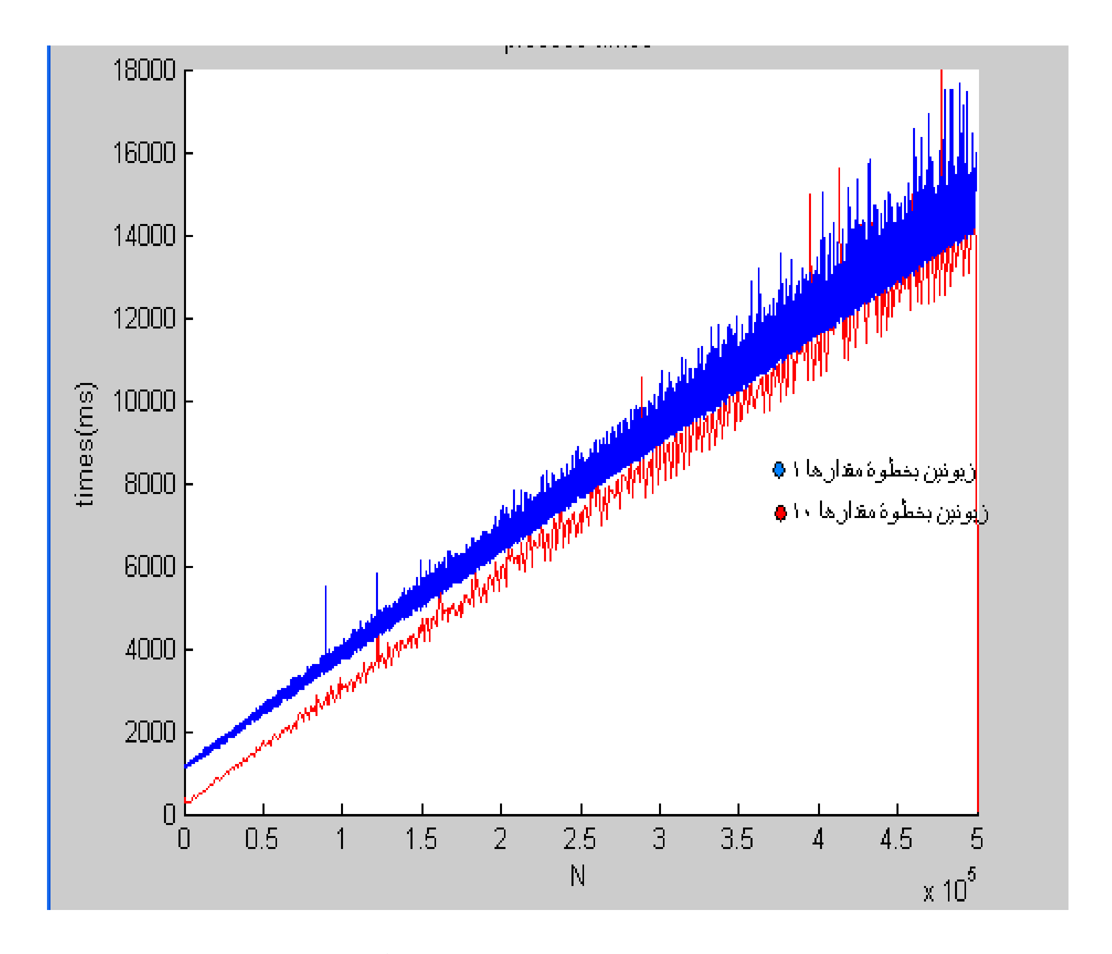
2- مقارنة نتائج توزيع العمل على ثلاثة أجهزة من أجل N=1..500000 وذلك على مرحلة واحدة ومن ثم على مرحلتين N=1..300000,N=300000..500000 وذلك لمعرفة تأثير إعادة تشغيل البرنامج على مرحلة التوزيع.

3- مقارنة نتئج جهاز واحد مع نتائج توزيع العمل على جهازين وصولاً إلى ثمانية أجهزة وذلك باستخدام شبكة جامعة حلب من أجل معرفة العوامل التي تؤثر على التوزيع.

7-2-1- المرحلة الأولى من التوزيع:

قمنا بمقارنة نتائج توزيع العمل على زبونين من أجل خطوة مقدارها 1 وأخرى مقدارها 10 .

عند تنفيذ التوزيع على حاسبين من أجل I=1..500000 وبخطوة مقدارها 1 ومن ثم تنفيذ التوزيع على حاسبين من أجل I=1..500000 وبخطوة مقدارها 10 حصلنا على المخططين الزمنيين التاليين الموضحين في الشكل (7-5):



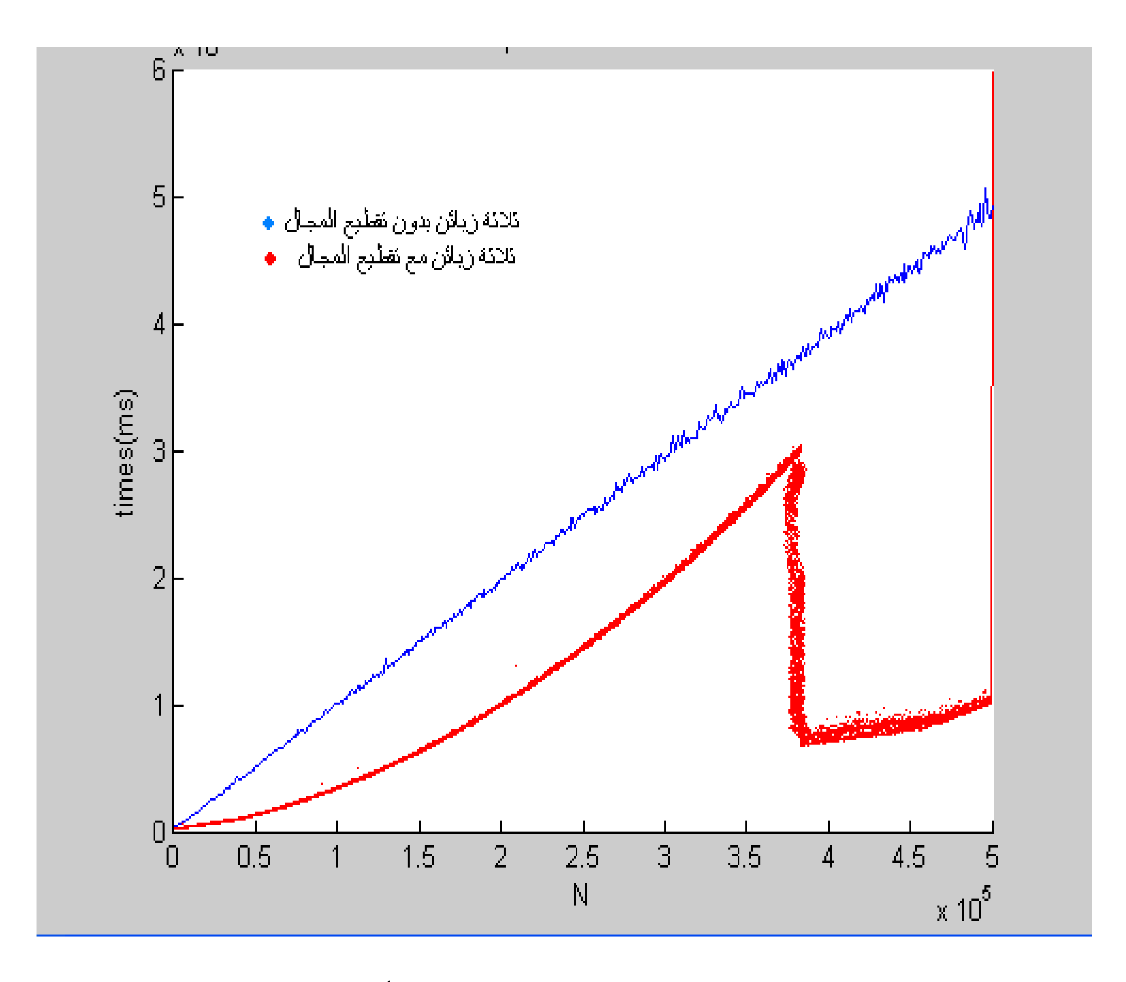
الشكل(7-5) نتائج التوزيع على زبونين بخطوة مقدارها 1 وأخرى مقدارها 10 .

نلاحظ أنه كلما صغرت الخطوة نحصل على أزمنة أكبر نسبياً فيما لو أخذنا عينات بخطوات كبيرة وذلك يعود إلى ازدياد استهلاك حجم الذاكرة المستخدمة في البرنامج لذلك يفضل أخذ عينات للتسريع في إيجاد النتائج.

7-2-2- المرحلة الثانية من التوزيع:

مقارنة نتائج التوزيع على ثلاث زبائن من أجل I=1..500000 متصلة ومرة أخرى من أجل I=1..300000 ومن ثم I=300000..500000 .

إن المنحنيين الزمنيين المعبرين عن التوزيع السابق موضحين بالشكل (7-6) التالي :



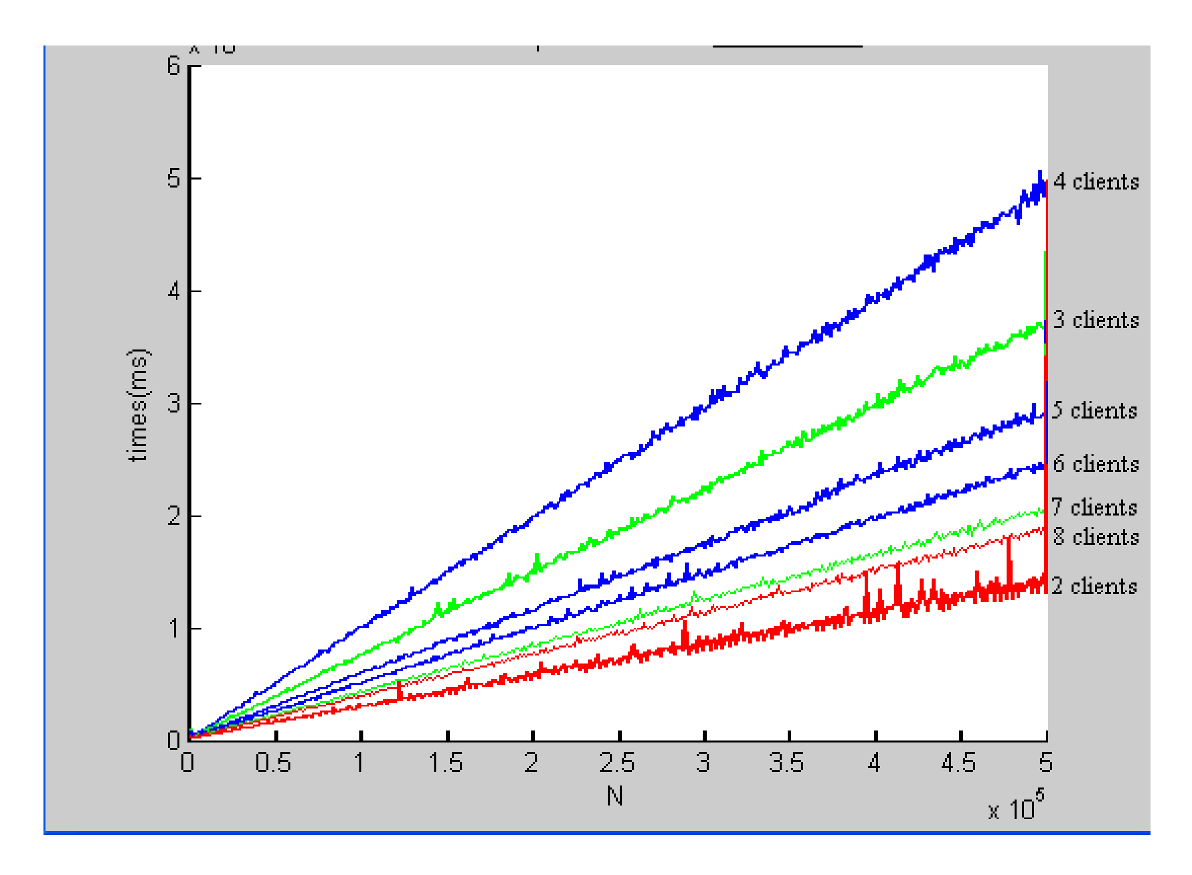
الشكل(7-6) نتائج التوزيع على ثلاثة زبائن مرة متصلة وأخرى متقطعة.

نلاحظ من الشكل السابق أن عملية تقطيع المجال لمرحلتين أو أكثر يؤدي إلى انخفاض الزمن المستهلك إلى حد كبير بسبب تفريغ الذواكر المستخدمة في البرنامج وهو من العوامل المهمة لذلك ينصح بتقطيع المجال الكلي إلى عدة مجالات كلية تزداد كلما زاد طول العلاقة .

كما أننا نلاحظ أن الزمن البدائي في المنحني الثاني أكبر من المنحني الأول وذلك لأننا قمنا بالتجربة ضمن الأسبوع الدراسي والضغط على شبكة جامعة حلب كبير أما تجربة ثلاثة زبائن مع تقطيع قمنا بها في نهاية الأسبوع والشبكة فارغة وغير مستخدمة من قبل الجامعة.

7-2-3- المرحلة الثالثة من التوزيع:

في هذه المرحلة تم توزيع العمل على زبونين وصولاً إلى ثمانية زبائن مع خطوة مقدارها 10 وبدون تقطيع ، ومنها حصلنا على المنحنيات البيانية التالية الموضحة في الشكل (7-7):



الشكل(7-7) نتائج التوزيع على زبونين وصولاً إلى ثمانية زبائن على شبكة خارجية.

من الشكل السابق نلاحظ أن مرحلة توزيع العمل على زبونين هي أسرع العمليات وذلك لأنه تم إجراء هذه التجربة في الليل وبدون وجود ضغط على شبكة جامعة حلب.

أما بالنسبة لتوزيع العمل على ثلاثة زبائن وأربعة نلاحظ وجود قفزة كبيرة في الزمن بدلاً من تقليل الزمن والسبب في ذلك أننا قمنا بإجراء التجربة ضمن الأسبوع وفي الفترة النهارية أي في فترة الذروة لاستخدام شبكة جامعة حلب.

ثم نلاحظ أنه اعتباراً من مرحلة خمسة زبائن وحتى مرحلة ثمانية زبائن تحسن للزمن وذلك بسبب أننا أخذنا النتائج في فترة العطلة الأسبوعية.

ومع ذلك هناك عوامل أخرى تؤثر على أداء الأداة المنطقية مثل جودة كابلات الشبكة والبروتوكول المستخدم في نقل البيانات.

7-2-4- خلاصة مرحلة التوزيع على شبكة جامعة حلب:

يمكن تلخيص النتائج السابقة بما يلي :

1- إن عملية التوزيع من أجل $i=1..N$ مع اتخاذ خطوات أكبر يساهم بشكل بسيط في تخفيض زمن حساب كل علاقة على حدة.

2- إن عملية التوزيع باستخدام مجالات متقطعة وإعادة تشغيل البرنامج في كل مرة يساهم في تخفيض الزمن بشكل كبير.

3- إن عملية التوزيع على مجموعة من الأجهزة يأخذ بعين الاعتبار الفترة التي تمت فيها من الأسبوع ، حيث نلاحظ تحسين المعالجة الموزعة لأداء الأداة المنطقية عندما تكون الشبكة غير مستخدمة في أعمال أخرى .

ملاحظة :

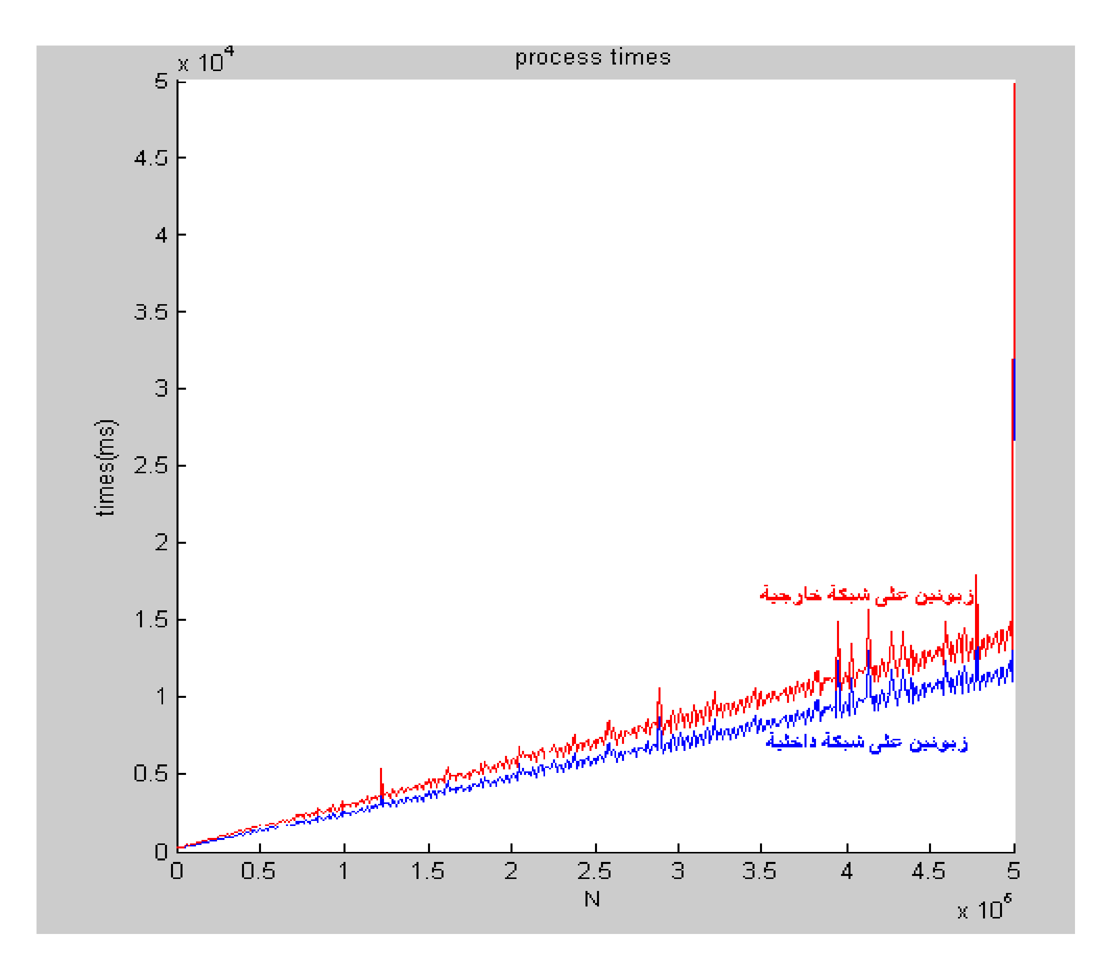
يجب الأخذ بعين الاعتبار العوامل المادية الأخرى عند التنفيذ مثل انقطاع التيار الكهربائي الفجائي ، أو تعطل الحواسيب الفجائي الذي قد يتسبب في إعادة إجراء كل تجربة أكثر من مرة حتى نستطيع الحصول على النتيجة المرجوة.

7-3- نتائج مراحل المعالجة الموزعة على شبكة داخلية ومقارنتها مع شبكة جامعة حلب:

قمنا بإعادة التوزيع على شبكة داخلية بدلاً من شبكة جامعة حلب لتوضيح تأثير نوع الشبكة على التوزيع واستنتاج الفائدة الحقيقية للتوزيع، استخدمنا في الشبكة الداخلية نفس الأجهزة التي استخدمناها في الشبكة الخارجية.

7-3-1- مقارنة توزيع العمل على زبونين بين شبكة داخلية وخارجية:

إن نتيجة توزيع العمل على زبونين موصولين على شبكة داخلية وآخرين موصولين على شبكة جامعة حلب موضحة بالشكل (7-8):

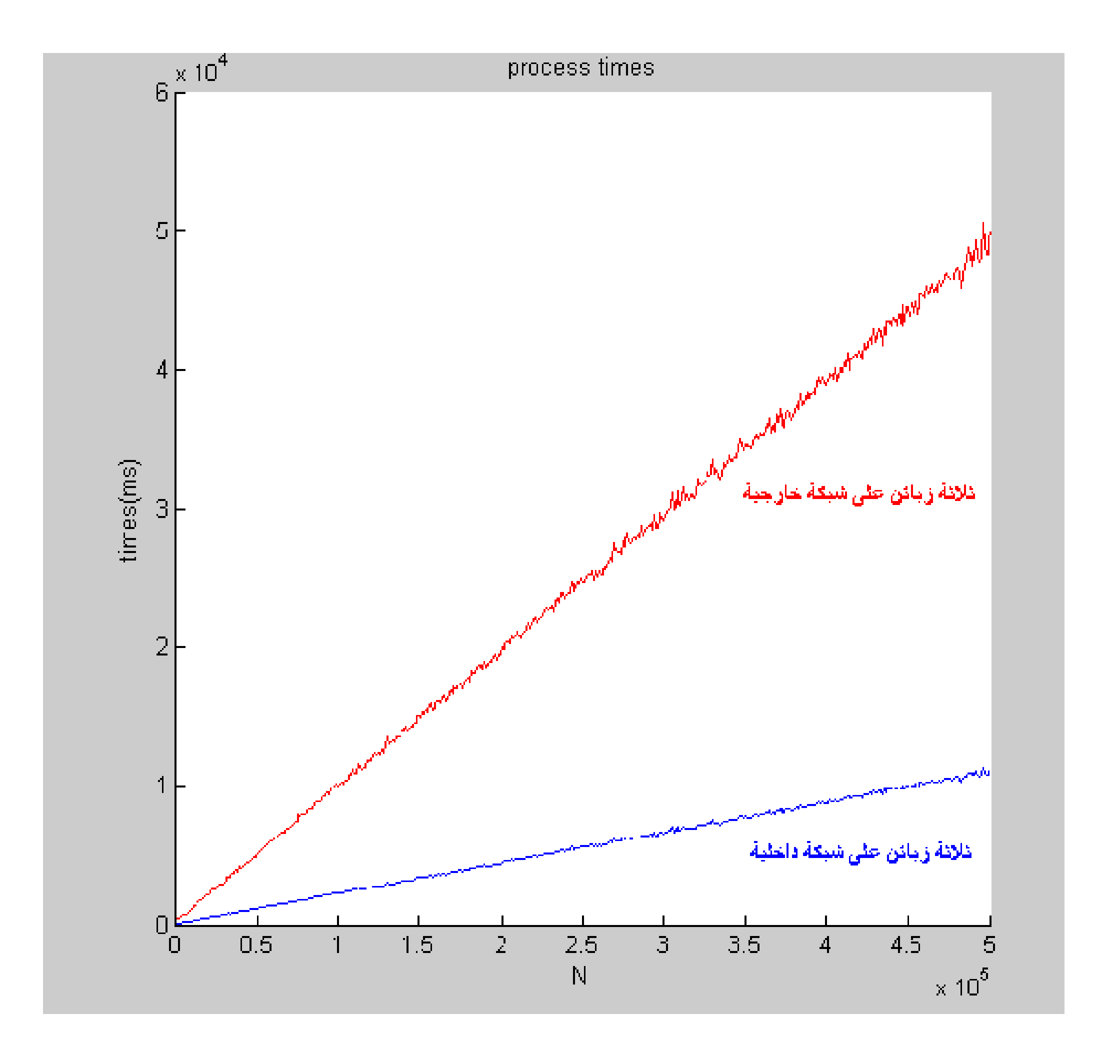


الشكل(7-8) نتيجة المقارنة لتوزيع العمل على شبكة داخلية وخارجية لزبونين.

من هذا الشكل نلاحظ أن هناك تناقص بسيط في الزمن عند تطبيق التوزيع على زبونين موصولين إلى شبكة داخلية مقارنةً مع نتيجة التوزيع على زبونين موصولين إلى شبكة خارجية. السبب في ذلك يرجع إلى أننا طبقنا التوزيع على زبونين موصولين إلى شبكة خارجية في عطلة نهاية الأسبوع أي بدون وجود ضغط على شبكة جامعة حلب.

7-3-2- مقارنة توزيع العمل على ثلاثة زبائن بين شبكة داخلية وخارجية:

إن نتيجة توزيع العمل على ثلاثة زبائن موصولين على شبكة داخلية وآخرين موصولين على شبكة جامعة حلب موضحة بالشكل (7-9):

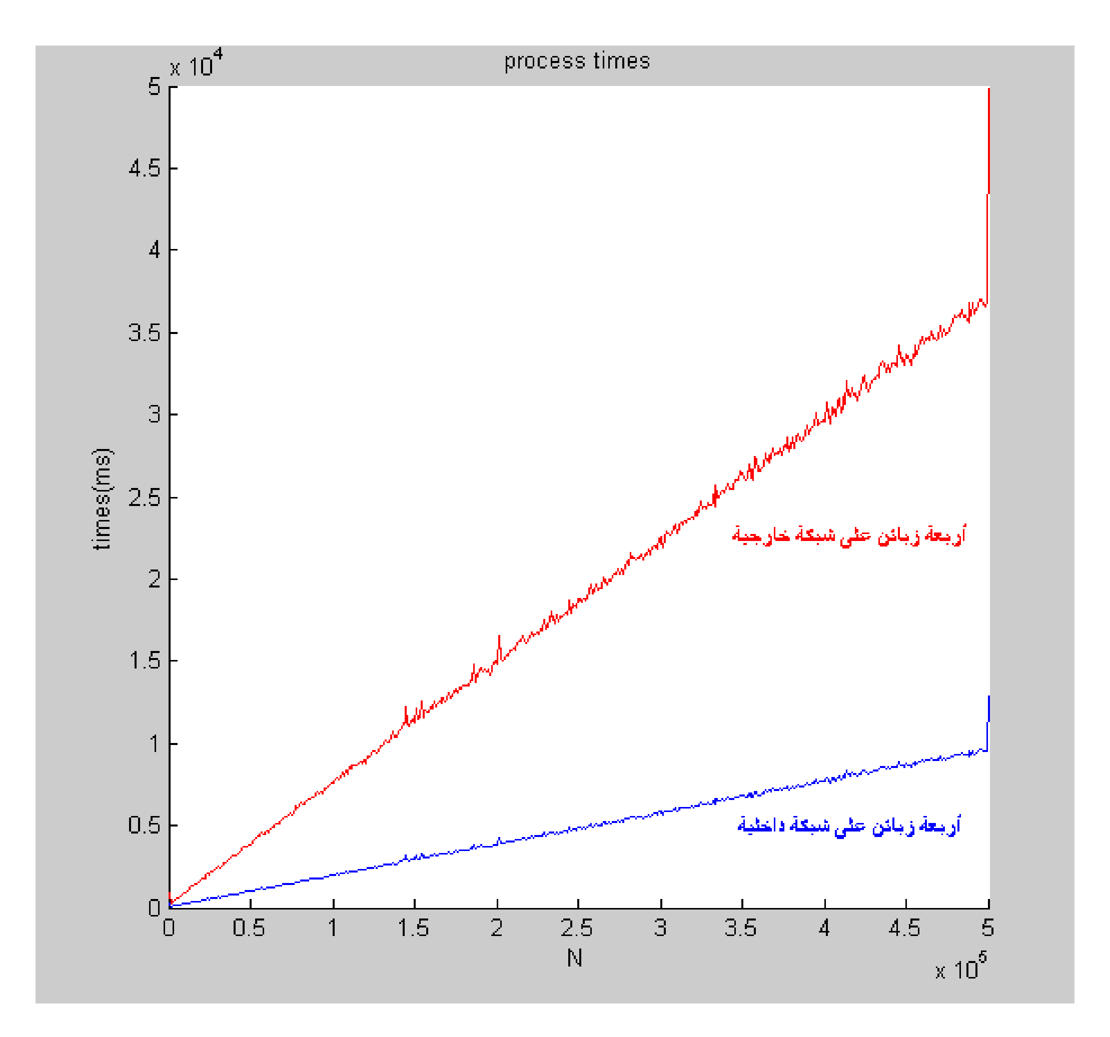


الشكل(7-9) نتيجة المقارنة لتوزيع العمل على شبكة داخلية وخارجية لثلاثة زبائن.

نلاحظ من هذا المخطط أن هناك تناقص كبير في الزمن عند استخدام شبكة داخلية، هذا التناقص الكبير يرجع إلى وجود ضغط كبير على الشبكة الخارجية وقت تنفيذ التجربة مما أدى إلى ضياع كبير في الزمن.

7-3-3- مقارنة توزيع العمل على أربعة زبائن بين شبكة داخلية وخارجية:

إن نتيجة توزيع العمل على أربعة زبائن موصولين على شبكة داخلية وآخرين موصولين على شبكة جامعة حلب موضحة بالشكل (7-10):

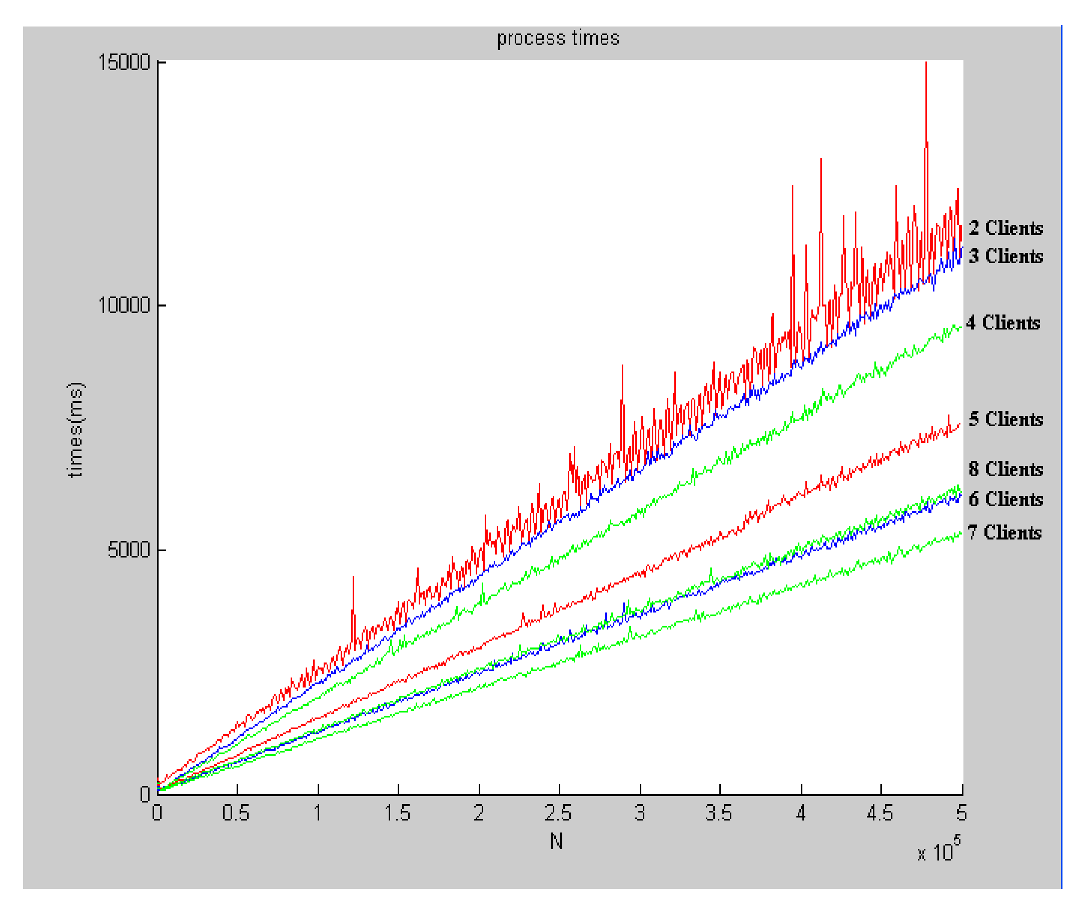


الشكل(7−10) نتيجة المقارنة لتوزيع العمل على شبكة داخلية وخارجية لأربعة زبائن.

نلاحظ من هذا المخطط أن هناك تناقص كبير في الزمن عند استخدام شبكة داخلية، هذا التناقص الكبير يرجع إلى وجود ضغط كبير على الشبكة الخارجية وقت تنفيذ التجربة مما أدى إلى ضياع كبير في الزمن كما هو الحال مع ثلاثة زبائن.

7−3−4− نتائج مرحلة التوزيع على شبكة داخلية:

في هذه المرحلة تم توزيع العمل على زبونين وصولاً إلى ثمانية زبائن موصولين إلى شبكة داخلية مع خطوة مقدارها 10 وبدون تقطيع ، ومنها حصلنا على المنحنيات البيانية التالية الموضحة في الشكل (7−11):



الشكل(7−11) نتائج لتوزيع على زبونين وصولاً إلى ثمانية زبائن على شبكة داخلية.

نلاحظ من هذا المخطط أنه كلما زاد عدد الزبائن يتناقص الزمن اعتباراً من زبونين وحتى سبعة زبائن ثم نلاحظ تزايد في الزمن عند الوصول لثمانية زبائن مما يعني أن عملية التخاطب بين المخدم والزبائن تستغرق زمناً أكبر منه على سبعة زبائن، وبالتالي عملية التوزيع على أكثر من سبعة زبائن هي عملية غير مجدية.

ومنه نستنتج أن استخدام شبكة داخلية في عملية التوزيع تعطي نتيجة أفضل من استخدام شبكة خارجية لأن الكابلات المستخدمة في توصيل الشبكة مخصصة فقط لعملية التخاطب بين المخدم والزبائن فقط في الشبكة الداخلية.

الفصل الثامن

واجهات البرامج التي تم استخدامها في مرحلة المعالجة الموزعة

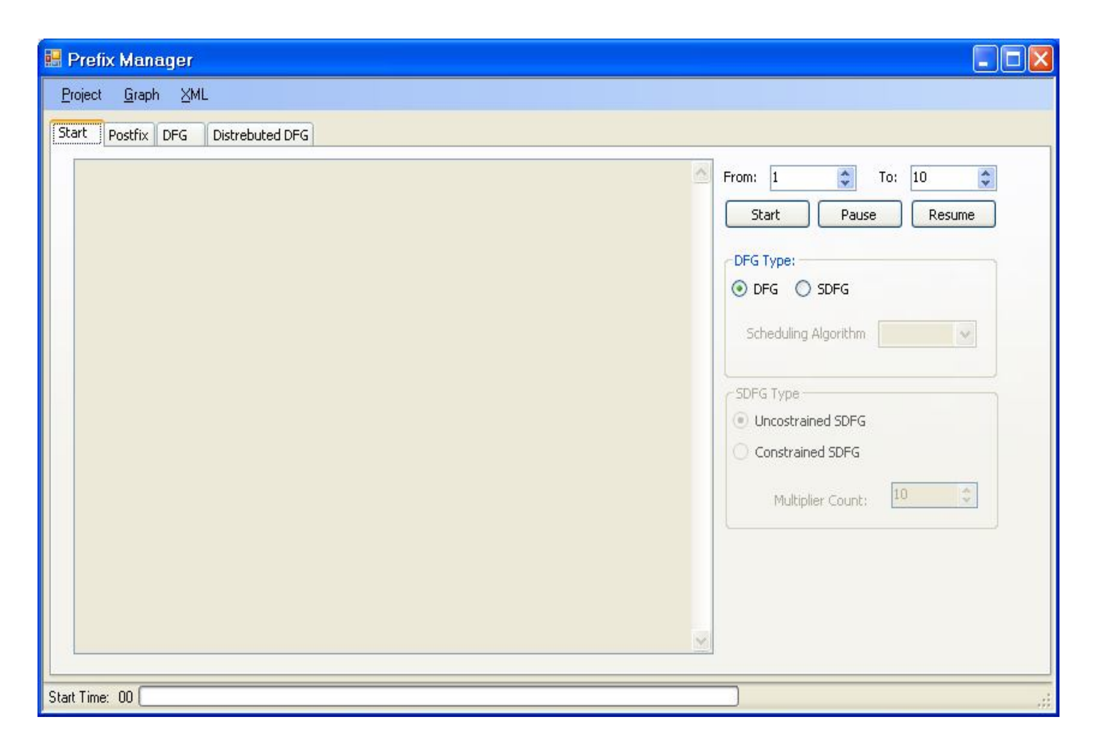
8-1- مقدمة:

في هذا الفصل سنستعرض العمل البرمجي الذي استخدمناه من أجل إيجاد نتائج الفصل السابق، استخدمنا في إنجاز هذا العمل البرمجي لغة الـ C# التي وجدنا أنها الأمثل في مرحلتي الترجمة والجدولة (انظر الفصل الرابع)، حيث قمنا بإنشاء برنامجين أحدهما يطبق على المخدم والآخر على الزبون . فيما يلي سنستعرض واجهات هذه البرامج بالتفصيل.

8-2- واجهات البرنامج في طرف المخدم:

نستعرض فيما يلي واجهات البرنامج المكتوب بلغة الـ C# المطبق على المخدم مع شرح مفصل لكل مكوناتها:

يبين الشكل (8-1) الواجهة الرئيسية لبرنامج المخدم :



الشكل (8-1) الواجهة الرئيسية لبرنامج المخدم.

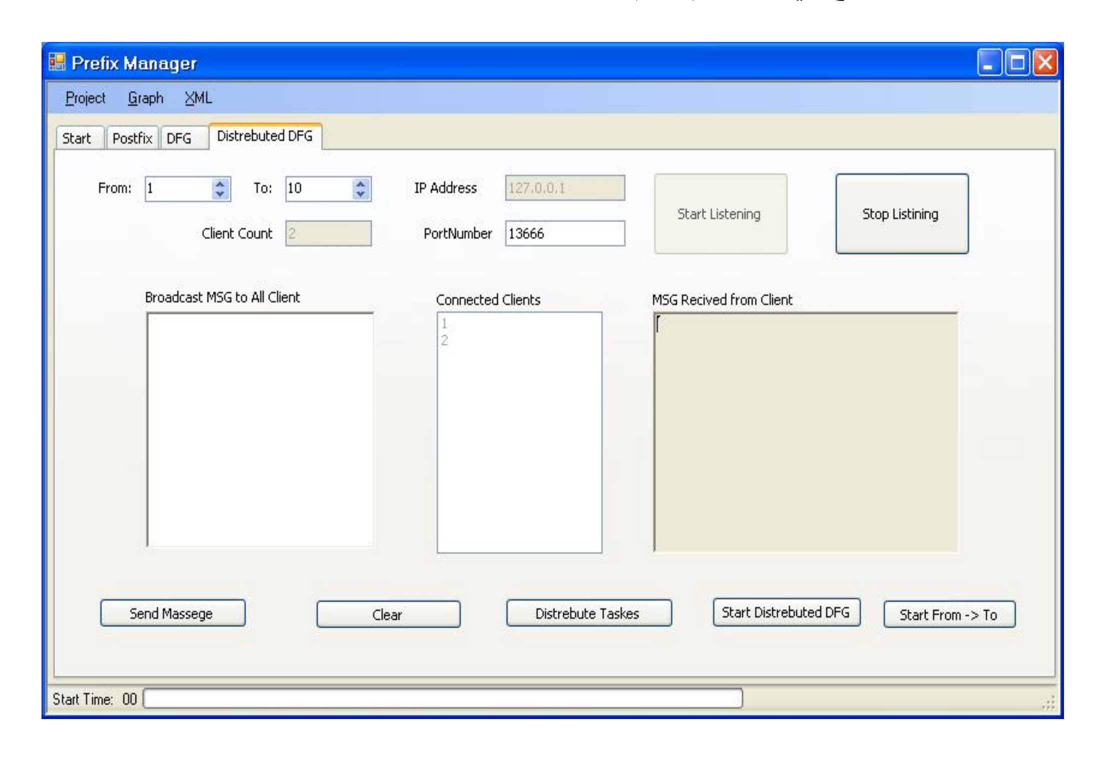
إن هذه الواجهة نفس الواجهة التي تم شرحها في الفصل الخامس والتي تحتوي على ثلاثة تبويبات:

1- التبويب Start : الذي نحدد فيه مرحلة الترجمة أو الجدولة مع تحديد نوع الجدولة بقيود أو بدون قيود بالإضافة للقيمة البدائية والنهائية للعلاقة.

2- التبويب Postfix : يظهر صيغة الـ Postfix للعلاقة بشكلها النهائي.

3- التبويب DFG : يظهر اللائحة المترابطة DFG أو SDFG .

تم إضافة تبويب رابع إلى هذه الواجهة هو التبويب Distributed DFG المسؤول عن عملية المعالجة الموزعة وهو موضح في الشكل (8-2):



الشكل (8-2) التبويب Distributed DFG لبرنامج المخدم.

في هذه الواجهة يتم تحديد القيمة البدائية والنهائية للعلاقة حيث سيقوم المخدم بتقسيم العلاقة بطولها البدائي بين الزبائن وينتهي بتقسيم العلاقة بطولها النهائي مع حساب أزمنة التخاطب وأزمنة التنفيذ. From: 1 To: 10

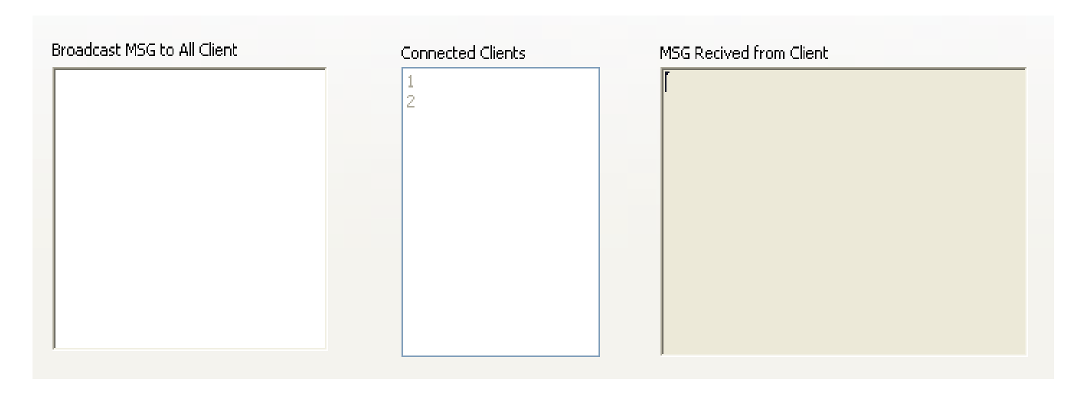
وتحتوي هذه الواجهة على عنوان الـ IP للمخدم ليتمكن الزبائن من الاتصال به ورقم الـ Port الذي سيتم التخاطب عبره، وهي من الأمور الهامة لنتمكن من استخدام بروتوكول TCP/IP .

IP Address 127.0.0.1

PortNumber 13666

كما تُظهر لنا عدد الزبائن المتصلة مع المخدم. Client Count 2

تحتوي أيضاً هذه الواجهة على ثلاثة نوافذ تسمح بالتخاطب مع الزبائن موضحة في الشكل (8-3):



الشكل(8-3) النوافذ التي تتيح التخاطب مع الزبائن.

- في النافذة Broadcast Msg to all Clients يتم إرسال رسالة عامة لكل الزبائن في نفس الوقت.
- في النافذة Connected Clients يتم إظهار أرقام الزبائن المتصلة مع المخدم.
- في النافذة MSG Received from Client يتم إظهار الرسائل المستقبلة من الزبائن مع تحديد الزبون مرسل الرسالة.

كما تحتوي هذه الواجهة على الأزرار التالية:

1- الزر Start Listening : Start Listening

يقوم هذا الزر بإعلام الزبائن بجاهزية المخدم للاتصال، حيث يقوم المخدم بالتنصت لمعرفة فيما إذا كان هناك زبون يرغب بالاتصال وعندها يقوم بفتح الجلسة.

Stop Listing

2- الزر Stop Listening :

يقوم هذا الزر بإغلاق الجلسة وانهاء عملية التخاطب مع الزبائن.

Send Massege

3- الزر Send Message :

يقوم بإرسال الرسالة العامة إلى كل الزبائن المتصلة.

Clear

4- الزر Clear :

يقوم بمسح نافذة الرسائل استعداداً لإرسال رسالة جديدة.

Distrebute Taskes

5- الزر Distribute Tasks:

يقوم بإيقاف البرنامج مؤقتاً لمتابعته فيما بعد من النقطة التي وصلنا إليها.

Start Distrebuted DFG

6- الزر Start Distributed DFG :

يتابع التنفيذ من النقطة التي وصلنا إليها.

Start From -> To

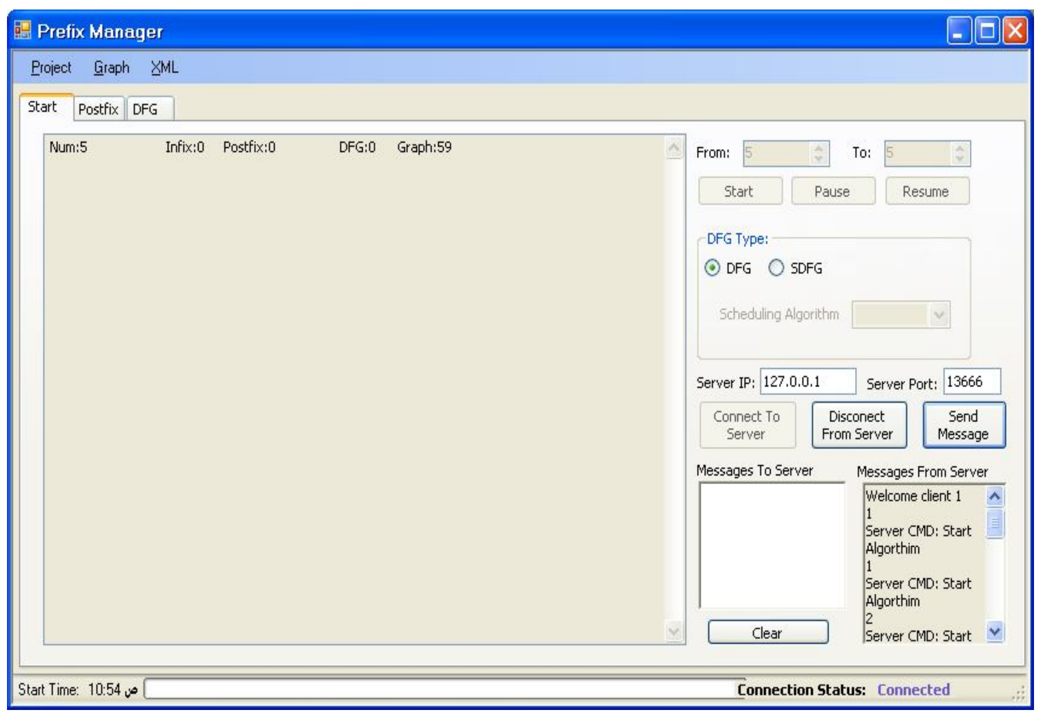
7- الزر Start from ->to:

يعطي الأمر ببدأ التنفيذ من القيمة البدائية إلى القيمة النهائية ليحسب أزمنة التنفيذ.

8-3- واجهات البرنامج في طرف الزبون:

نستعرض فيما يلي واجهات البرنامج المكتوب بلغة الـ #C المطبق على الزبون مع شرح مفصل لكل مكوناتها:

يبين الشكل (8-4) الواجهة الرئيسية لبرنامج الزبون :

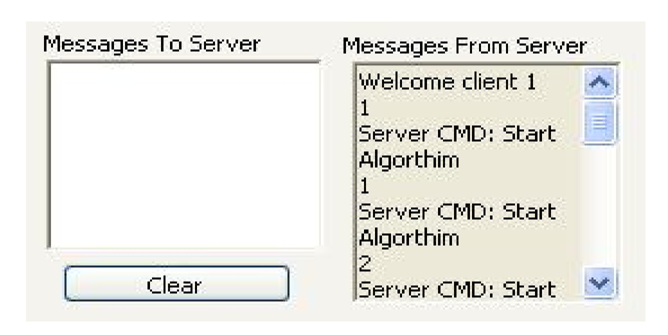


الشكل (8-4) الواجهة الرئيسية لبرنامج الزبون.

هذه الواجهة تماثل واجهة برنامج الـ C# التي استخدمناها في مرحلتي الجدولة والترجمة ولكن تم إضافة عنوان المخدم الذي سيقوم الزبون بالاتصال به إلى جانب عنوان الـ Port الذي سيتم تبادل المعلومات عبره باستخدام بروتوكول TCP/IP .



وتم إضافة نافذتين إلى الواجهة موضحتين في الشكل (8-5):



الشكل(8-5) النوافذ التي تتيح التخاطب مع المخدم.

- في النافذة Messages to Server يتم إرسال رسالة معينة إلى المخدم.
- في النافذة Messages from Server يتم إظهار الرسائل المستقبلة من المخدم.

كما تحتوي هذه الواجهة على الأزرار الإضافية التالية:

1- الزر Connect to Server : Connect To Server

يقوم هذا الزر بإرسال رسالة إلى المخدم تحتوي رغبة الزبون في إنشاء اتصال مع المخدم عندما يكون المخدم في حالة تنصت.

2- الزر Disconnect from Server : Disconnect From Server

يقوم بقطع الاتصال بين المخدم والزبون وإغلاق الجلسة.

3- الزر Send Message : Send Message

يقوم بإرسال الرسالة الموجودة في نافذة Message to Server إلى المخدم.

4- الزر Clear : Clear

يقوم بمسح نافذة Message to Sever تمهيداً لإرسال رسالة جديدة.

الفصل التاسع
النتائج والنظرة المستقبلية

9-1- ملخص النتائج:

يمكن تلخيص النتائج على الشكل التالي:

1- من مرحلة مقارنة لغات البرمجة الثلاثة : C#-Delphi-Java نجد أن اللغة الأنسب لمتابعة العمل في بناء الأداة المنطقية هي لغة الـ C# لما تملكه من خصائص مميزة في التعامل مع عناصر الرسم بالإضافة إلى أزمنة التنفيذ الجيدة مقارنةً مع غيرها، وكمية المعطيات الكبيرة نسبياً القابلة للمعالجة.

2- من مرحلة المعالجة الموزعة على شبكة خارجية حصلنا على النتائج التالية:

- إن عملية التوزيع من أجل i=1..N مع اتخاذ خطوات أكبر يساهم بشكل بسيط في تخفيض زمن حساب كل علاقة على حدة.

- إن عملية التوزيع باستخدام مجالات متقطعة وإعادة تشغيل البرنامج في كل مرة يساهم في تخفيض الزمن بشكل كبير.

- إن عملية التوزيع على مجموعة من الأجهزة يأخذ بعين الاعتبار الفترة التي تمت فيها من الأسبوع ، حيث نلاحظ تحسين المعالجة الموزعة لأداء الأداة المنطقية عندما تكون الشبكة غير مستخدمة في أعمال أخرى .

3- من مرحلة المعالجة الموزعة على شبكة داخلية وجدنا أنه كلما زاد عدد الزبائن يتناقص الزمن اعتباراً من زبونين وحتى سبعة زبائن ثم نلاحظ تزايد في الزمن عند الوصول لثمانية زبائن مما يعني أن عملية التخاطب بين المخدم والزبائن تستغرق زمناً أكبر منه على سبعة زبائن، وبالتالي عملية التوزيع على أكثر من سبعة زبائن هي عملية غير مجدية.

4- إن استخدام شبكة داخلية في عملية التوزيع تعطي نتيجة أفضل من استخدام شبكة خارجية لأن الكابلات المستخدمة في توصيل الشبكة مخصصة فقط لعملية التخاطب بين المخدم والزبائن فقط في الشبكة الداخلية.

9-2- النظرة المستقبلية:

باعتبار أن هذا البحث يدخل في خطة قسم هندسة الحواسيب لتطوير أدوات تصميم للنظم الرقمية فإن المرحلة التالية يجب أن تركز على متابعة بناء النظام في مرحلة البناء عالي المستوى وصولاً إلى تنفيذ النظام في مستوى RTL على شرائح قابلة للبرمجة FPGA ثم الانتقال إلى مرحلة البناء منخفض المستوى LLS وصولاً إلى التنفيذ الفيزيائي للدارة التكاملية IC .

ونأمل لهذا المشروع أن يتابع الرقي في مسار التطور للوصول إلى الآمال المرجوة منه، وأن يكون خطوة البداية للتوجه نحو مجالات التصميم في عالمنا العربي.

الملحقات

10-1 الشيفرة المصدرية لمرحلتي الترجمة والجدولة بلغة C# :

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Threading;
using System.Diagnostics;
using PrefixManager.Classes;

namespace PrefixManager
{
    public partial class Form1 : Form
    {
        public static Dictionary<int, Period> PeriodTable = new
Dictionary<int, Period>();
        public static Dictionary<int, GeneralFactory> DfgTable;
        Thread thread;
        private DateTime startTime;
        private SchedulingAlgorithm algorithm;

        public Form1()
        {
            InitializeComponent();

            Control.CheckForIllegalCrossThreadCalls = false;
        }

        private void Calc()
        {
            txtResults.Clear();
            listView1.Items.Clear();

            Period p;
            PeriodTable.Clear();
            int From = Convert.ToInt32(numFrom.Value);
            int To = Convert.ToInt32(numTo.Value);
            string temp2 = string.Empty, temp1 = string.Empty;
            int stepCount = To - From + 1;


            startTime = DateTime.Now;
            tsStartTime.Text = startTime.ToShortTimeString();
            lblElapsedTime.Text = string.Empty;

            Stopwatch w = new Stopwatch();
            DfgTable = new Dictionary<int, GeneralFactory>(To);

            tsProgressBar.Value = 0;

            for (int i = From; i <= To; i++)
            {
                p = new Period();

                #region Infix Creation
                //Infix Creation
                InfixExpression infix = new InfixExpression(i);

                w.Start();
```

```
                infix.Create();
                w.Stop();

                p.InfixCreation = w.ElapsedMilliseconds;
                #endregion

                w.Reset();

                #region Postfix Creation
                //Postfix Creation
                PostfixExpression postfix = new
PostfixExpression(infix.InfixExp);
                w.Start();
                postfix.Create();
                temp1 = postfix.PostfixExp;
                w.Stop();

                p.PostfixCreation = w.ElapsedMilliseconds;

                #endregion

                w.Reset();

                #region DFG Creation
                //DFG Creation
                //SdfgFactory factory = new
SdfgFactory(postfix.PostfixExp, algorithm);
                GeneralFactory factory;
                if (rbDfg.Checked)
                    factory = new DfgFactory(postfix.PostfixExp);
                else
                    factory = new SdfgFactory(postfix.PostfixExp,
algorithm);
                w.Start();
                factory.Create();
                w.Stop();
                temp2 = factory.DfgExpression;
                DfgTable.Add(i, factory);

                p.DfgCreation = w.ElapsedMilliseconds;

                //DfgTable[i - 1] = factory;
                #endregion

                w.Reset();

                #region DFG Graph Creation
                //DFG Graph Creation
                DFGGraph g = new DFGGraph(factory.DfgList,
algorithm);
                w.Start();
                g.Draw();
                w.Stop();
                g.Dispose();

                p.DfgGraphCreation = w.ElapsedMilliseconds;
                #endregion

                w.Reset();
```

```
                infix.Dispose();

                //Draw Period


                PeriodTable[i] = p;


txtResults.AppendText(string.Format("Num:{0}\t\tInfix:{1}\tPostfix:{2
}\t\tDFG:{3}\tGraph:{4}\n",
                    i,
                    PeriodTable[i].InfixCreation,
                    PeriodTable[i].PostfixCreation,
                    PeriodTable[i].DfgCreation,
                    PeriodTable[i].DfgGraphCreation));

                if (i % 2000 == 0)
                {
                    XMLGraph.WriteToFile(PeriodTable,
string.Format("{0}.xml", i.ToString()));
                }

                //tsProgressBar.Value += stepCount / 100;
            }

            txtPostfixOutput.Text = temp1 + "\n";
            txtDFGOutput.Text = temp2 + "\n";

            timer1.Enabled = false;
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            if (PeriodTable != null)
                PeriodTable.Clear();
            if (DfgTable != null)
                DfgTable.Clear();

            thread = new Thread(new ThreadStart(Calc));

            thread.Start();

        }

        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            if (thread != null)
                thread.Abort();
        }

        private void closeToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Application.Exit();
        }

        private void graphComparerToolStripMenuItem_Click(object
sender, EventArgs e)
```

```
        {
            XmlGraphForm form = new XmlGraphForm();

            form.Show();
        }

        private void btnPauses_Click(object sender, EventArgs e)
        {
            if (thread != null)
                thread.Suspend();
        }

        private void btnResume_Click(object sender, EventArgs e)
        {
            if (thread != null)
                thread.Resume();
        }

        private void generateTimeGraphToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            PostfixGraph gr = new PostfixGraph();
            gr.Show();
        }

        private void dFGGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            DFGGraph graph = new DFGGraph(DfgTable[DfgTable.Count -
1].DfgList, algorithm);
            graph.Show();
        }

        private void generateXMLToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                XMLGraph.WriteToFile(PeriodTable,
saveFileDialog1.FileName);
            }
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            TimeSpan t = DateTime.Now.Subtract(startTime);
            lblElapsedTime.Text = string.Format("{0}:{1}:{2}",
t.Hours, t.Minutes, t.Seconds);

        }

        private void rbSdfg_CheckedChanged(object sender, EventArgs
e)
        {
            if (rbSdfg.Checked == true)
            {
                pnlSdfg.Enabled = true;
                cbAlgorithm.SelectedIndex = 0;
            }
```

```
            else
                pnlSdfg.Enabled = false;
        }

        private void cbAlgorithm_SelectedIndexChanged(object sender,
EventArgs e)
        {
            algorithm =

(SchedulingAlgorithm)Enum.Parse(typeof(SchedulingAlgorithm),
                cbAlgorithm.SelectedItem.ToString());
        }
    }
}
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;
using PrefixManager.Classes;

namespace PrefixManager
{
    public partial class DFGGraph : Form
    {
        #region Variables
        DFG[] list;
        private const int HORIZONTAL_SPACING = 50;
        private const int VERTICAL_SPACING = 100;
        private const int CIRCLE_WIDTH = 25;
        private readonly Brush OPERATION_CIRCLE_BRUSH = new
SolidBrush(Color.Blue);
        private readonly Brush UNARY_CIRCLE_BRUSH = new
SolidBrush(Color.Red);
        private readonly Brush BINARY_CIRCLE_BRUSH = new
SolidBrush(Color.Green);
        private readonly Pen LINE_PEN = new Pen(Color.Magenta);

        private delegate void DrawDelegate();
        private DrawDelegate drawMethod = null;
        private SchedulingAlgorithm algorithm;

        private SDFG s = null;
        #endregion

        #region Constructor
        public DFGGraph(DFG[] list, SchedulingAlgorithm algorithm)
        {
            InitializeComponent();

            this.list = list;

            LINE_PEN.Width = 4;
            LINE_PEN.StartCap = LineCap.ArrowAnchor;

            if (list is SDFG[])
            {
                this.Text = "SDFG Graph";

                this.algorithm = algorithm;
```

```
            switch (algorithm)
            {
                case SchedulingAlgorithm.ASAP:
                    drawMethod = DrawASAP;
                    break;
                case SchedulingAlgorithm.ALAP:
                    drawMethod = DrawALAP;
                    break;
            }


        }
        else if (list is DFG[])
        {
            drawMethod = DrawDFG;
        }
    }
    #endregion

    #region Draw
    public void Draw()
    {
        drawMethod.Invoke();
    }
    #endregion

    #region Draw ALAP
    private void DrawALAP()
    {
        if (list == null)
            return;

        Image img = new Bitmap(2000, 2000);
        Graphics g = Graphics.FromImage(img);
        pictureBox1.Image = img;
        g.SmoothingMode = SmoothingMode.AntiAlias;


        int x = 0, y = VERTICAL_SPACING;

        for (int i = 0; i < list.Length; i++)
        {
            s = list[i] as SDFG;

            if (s == null)
                break;

            if (s.State == DFGState.operation)
            {

                int m = VERTICAL_SPACING * s.Level;

                y += VERTICAL_SPACING;
                g.FillEllipse(OPERATION_CIRCLE_BRUSH,
                    x,
                    m,
                    CIRCLE_WIDTH, CIRCLE_WIDTH);
```

```
                    //Draw Lines
                    g.DrawLine(LINE_PEN,
                        x + CIRCLE_WIDTH / 2,
                        m,
                        x + CIRCLE_WIDTH / 2,
                        m - VERTICAL_SPACING + CIRCLE_WIDTH);
                    g.DrawLine(LINE_PEN,
                        x + CIRCLE_WIDTH / 3,
                        m,
                        x - 2 * HORIZONTAL_SPACING + 2 * CIRCLE_WIDTH
/ 3,
                        m - VERTICAL_SPACING + CIRCLE_WIDTH);

                    g.DrawString(s.Operation.ToString(),
                        new Font("Tahoma", 14f), Brushes.Black,
                        x + 3,
                        m);

                    g.DrawRectangle(Pens.Black, x + CIRCLE_WIDTH + 8,
m, 2 * CIRCLE_WIDTH, CIRCLE_WIDTH);
                    g.DrawLine(Pens.Black, x + (2 * CIRCLE_WIDTH) +
8, m, x + (2 * CIRCLE_WIDTH) + 8, m + CIRCLE_WIDTH);
                    g.DrawString(s.Level.ToString(),
                                            new Font("Veradana",
14f), Brushes.Black,
                                            x + CIRCLE_WIDTH + 12,
                                            m);
                    g.DrawString(s.ControlStep.ToString(),
                                            new Font("Veradana",
14f), Brushes.Black,
                                            x + 2 * CIRCLE_WIDTH +
12,
                                            m);

                }
                else if (s.State == DFGState.unary)
                {
                    x += HORIZONTAL_SPACING;
                    g.FillEllipse(UNARY_CIRCLE_BRUSH, x, y,
CIRCLE_WIDTH, CIRCLE_WIDTH);
                }
                else if (s.State == DFGState.binary)
                {
                    int m = VERTICAL_SPACING * s.ControlStep;

                    x += HORIZONTAL_SPACING;
                    x += HORIZONTAL_SPACING;
                    g.FillEllipse(BINARY_CIRCLE_BRUSH,
                        x,
                        m,
                        CIRCLE_WIDTH,
                        CIRCLE_WIDTH);

                    g.DrawString(s.Operation.ToString(),
                        new Font("Tahoma", 14f), Brushes.Black,
                        x + 3,
                        m);
```

```
                    g.DrawRectangle(Pens.Black, x + CIRCLE_WIDTH + 8,
m, 2 * CIRCLE_WIDTH, CIRCLE_WIDTH);
                    g.DrawLine(Pens.Black, x + (2 * CIRCLE_WIDTH) +
8, m, x + (2 * CIRCLE_WIDTH) + 8, m + CIRCLE_WIDTH);
                    g.DrawString(s.Level.ToString(),
                                            new Font("Veradana",
14f), Brushes.Black,
                                            x + CIRCLE_WIDTH + 12,
                                            m);
                    g.DrawString(s.ControlStep.ToString(),
                                            new Font("Veradana",
14f), Brushes.Black,
                                            x + 2 * CIRCLE_WIDTH +
12,
                                            m);

                }
            }
        }
        #endregion

        #region Draw ASAP
        private void DrawASAP()
        {
            if (list == null)
                return;

            Image img = new Bitmap(2000, 2000);
            Graphics g = Graphics.FromImage(img);
            pictureBox1.Image = img;
            g.SmoothingMode = SmoothingMode.AntiAlias;


            int x = 0, y = VERTICAL_SPACING;

            for (int i = 0; i < list.Length; i++)
            {
                s = list[i] as SDFG;

                if (s == null)
                    break;

                if (s.State == DFGState.operation)
                {
                    y += VERTICAL_SPACING;
                    g.FillEllipse(OPERATION_CIRCLE_BRUSH,
                        x, y,
                        CIRCLE_WIDTH, CIRCLE_WIDTH);


                    SDFG prevLeft, prevRight;
                    prevRight = list[s.VL - 1] as SDFG;
                    prevLeft = list[s.VR - 1] as SDFG;

                    int m = VERTICAL_SPACING * prevLeft.ControlStep;

                    //Draw Lines
                    g.DrawLine(LINE_PEN,
                        x + CIRCLE_WIDTH / 2,
```

```
                                y,
                                x + CIRCLE_WIDTH / 2,
                                m + CIRCLE_WIDTH);
                            g.DrawLine(LINE_PEN,
                                x + CIRCLE_WIDTH / 3,
                                y,
                                x - 2 * HORIZONTAL_SPACING + 2 * CIRCLE_WIDTH
/ 3,
                                y - VERTICAL_SPACING + CIRCLE_WIDTH);

                            g.DrawString(s.Operation.ToString(),
                                new Font("Tahoma", 14f), Brushes.Black,
                                x + 3, y);

                            g.DrawRectangle(Pens.Black, x + CIRCLE_WIDTH + 8,
y, 2 * CIRCLE_WIDTH, CIRCLE_WIDTH);
                            g.DrawLine(Pens.Black, x + (2 * CIRCLE_WIDTH) +
8, y, x + (2 * CIRCLE_WIDTH) + 8, y + CIRCLE_WIDTH);
                            g.DrawString(s.Level.ToString(),
                                                    new Font("Veradana",
14f), Brushes.Black,
                                                    x + CIRCLE_WIDTH + 12,
                                                    y);
                            g.DrawString(s.ControlStep.ToString(),
                                                    new Font("Veradana",
14f), Brushes.Black,
                                                    x + 2 * CIRCLE_WIDTH +
12,
                                                    y);


                        }
                        else if (s.State == DFGState.unary)
                        {
                            x += HORIZONTAL_SPACING;
                            g.FillEllipse(UNARY_CIRCLE_BRUSH, x, y,
CIRCLE_WIDTH, CIRCLE_WIDTH);
                        }
                        else if (s.State == DFGState.binary)
                        {

                            int m = VERTICAL_SPACING * s.ControlStep;

                            x += HORIZONTAL_SPACING;
                            x += HORIZONTAL_SPACING;
                            g.FillEllipse(BINARY_CIRCLE_BRUSH,
                                x,
                                m,
                                CIRCLE_WIDTH,
                                CIRCLE_WIDTH);

                            g.DrawString(s.Operation.ToString(),
                                new Font("Tahoma", 14f), Brushes.Black,
                                x + 3,
                                m);

                            g.DrawRectangle(Pens.Black, x + CIRCLE_WIDTH + 8,
m, 2 * CIRCLE_WIDTH, CIRCLE_WIDTH);
```

```
                    g.DrawLine(Pens.Black, x + (2 * CIRCLE_WIDTH) +
8, m, x + (2 * CIRCLE_WIDTH) + 8, m + CIRCLE_WIDTH);
                    g.DrawString(s.Level.ToString(),
                                                new Font("Veradana",
14f), Brushes.Black,
                                                x + CIRCLE_WIDTH + 12,
                                                m);
                    g.DrawString(s.ControlStep.ToString(),
                                                new Font("Veradana",
14f), Brushes.Black,
                                                x + 2 * CIRCLE_WIDTH +
12,
                                                m);

                }
            }
        }
        #endregion

        #region Draw DFG
        private void DrawDFG()
        {
            if (list == null)
                return;

            Image img = new Bitmap(2000, 2000);
            Graphics g = Graphics.FromImage(img);
            pictureBox1.Image = img;
            g.SmoothingMode = SmoothingMode.AntiAlias;


            int x = 0, y = 90;

            for (int i = 0; i < list.Length; i++)
            {
                if (list[i] == null)
                    break;

                if (list[i].State == DFGState.operation)
                {
                    y += VERTICAL_SPACING;
                    g.FillEllipse(OPERATION_CIRCLE_BRUSH, x, y,
CIRCLE_WIDTH, CIRCLE_WIDTH);

                    //Draw Lines
                    g.DrawLine(LINE_PEN, x + CIRCLE_WIDTH / 2, y, x +
CIRCLE_WIDTH / 2, y - VERTICAL_SPACING + CIRCLE_WIDTH);
                    g.DrawLine(LINE_PEN,
                        x + CIRCLE_WIDTH / 3,
                        y,
                        x - 2 * HORIZONTAL_SPACING + 2 * CIRCLE_WIDTH
/ 3,
                        y - VERTICAL_SPACING + CIRCLE_WIDTH);
                }
                else if (list[i].State == DFGState.unary)
                {
                    x += HORIZONTAL_SPACING;
                    g.FillEllipse(UNARY_CIRCLE_BRUSH, x, y,
CIRCLE_WIDTH, CIRCLE_WIDTH);
```

```
                }
                else if (list[i].State == DFGState.binary)
                {
                    x += HORIZONTAL_SPACING;
                    x += HORIZONTAL_SPACING;
                    g.FillEllipse(BINARY_CIRCLE_BRUSH, x, y,
CIRCLE_WIDTH, CIRCLE_WIDTH);
                }

                g.DrawString(list[i].Operation.ToString(), new
Font("Tahoma", 14f), Brushes.Black, x + 3, y);

            }
        }
        #endregion

        #region Form Load Event
        private void DFGGraph_Load(object sender, System.EventArgs e)
        {
            Draw();
        }
        #endregion
    }
}
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using PrefixManager.Classes;

namespace PrefixManager
{
    public partial class PostfixGraph : Form
    {
        PointF zeroCoordinates;
        const int xShift = 50;
        const int yShift = 50;
        readonly Color coordinatesColor = Color.Black;
        private float horizontalShift = 1;
        private float verticalShift = 5;
        Pen coordinatesPen;
        Pen myPen = new Pen(Color.Red);
        Period p1, p2;
        Graphics g;

        PointF po1 = new PointF(), po2 = new PointF();

        public PostfixGraph()
        {
            InitializeComponent();

            coordinatesPen = new Pen(coordinatesColor, 3f);
            coordinatesPen.EndCap = LineCap.ArrowAnchor;
            myPen.DashStyle = DashStyle.Dash;


        }

        private void pbGraph_Paint(object sender, PaintEventArgs e)
```

```
        {

            g = e.Graphics;
            g.SmoothingMode = SmoothingMode.AntiAlias;

            // if (pbGraph.ClientRectangle.Width > Form1.table.Count)
            horizontalShift = (float)(pbGraph.ClientRectangle.Width -
xShift - zeroCoordinates.X) / Form1.PeriodTable.Count;


            //Draw Coordinates
            g.DrawLine(coordinatesPen, zeroCoordinates, new
PointF(pbGraph.ClientRectangle.Width - xShift, zeroCoordinates.Y));
            g.DrawLine(coordinatesPen, zeroCoordinates, new
PointF(zeroCoordinates.X, yShift));

            int counter = Form1.PeriodTable.Count / 10;

            for (int i = 1; i < Form1.PeriodTable.Count; i++)
            {
                //    pbGraph.Height += 100;

                if (i % counter == 0)
                {
                    g.DrawLine(Pens.Black, zeroCoordinates.X + i *
horizontalShift, zeroCoordinates.Y + 5,
                        zeroCoordinates.X + i * horizontalShift,
zeroCoordinates.Y - 5);
                    g.DrawString(i.ToString(),
SystemFonts.DialogFont, Brushes.Black,
                        new PointF(zeroCoordinates.X + i *
horizontalShift, zeroCoordinates.Y + 15));
                }

                p1 = Form1.PeriodTable[i];
                p2 = Form1.PeriodTable[i + 1];


                po1.X = i * horizontalShift + xShift;
                po1.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p1.DfgGraphCreation * verticalShift);

                po2.X = (i + 1) * horizontalShift + xShift;
                po2.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p2.DfgGraphCreation * verticalShift);

                g.DrawLine(Pens.Black, po1, po2);

                po1.X = i * horizontalShift + xShift;
                po1.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p1.PostfixCreation * verticalShift);

                po2.X = (i + 1) * horizontalShift + xShift;
                po2.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p2.PostfixCreation * verticalShift);

                g.DrawLine(Pens.Red, po1, po2);

                po1.X = i * horizontalShift + xShift;
```

```
                    po1.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p1.DfgCreation * verticalShift);

                    po2.X = (i + 1) * horizontalShift + xShift;
                    po2.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p2.DfgCreation * verticalShift);

                    g.DrawLine(Pens.Green, po1, po2);

                    po1.X = i * horizontalShift + xShift;
                    po1.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p1.TotalTime * verticalShift);

                    po2.X = (i + 1) * horizontalShift + xShift;
                    po2.Y = (int)(pbGraph.ClientRectangle.Height - yShift
- p2.TotalTime * verticalShift);


                    g.DrawLine(Pens.Blue, po1, po2);
                }

                //Draw Red lines
                g.DrawLine(myPen, new PointF(zeroCoordinates.X, po2.Y),
po2);
                g.DrawLine(myPen, new PointF(po2.X, zeroCoordinates.Y),
po2);

                //Draw strings
                g.DrawString(Form1.PeriodTable.Count.ToString(),
SystemFonts.DialogFont, Brushes.Black, new
PointF(pbGraph.ClientRectangle.Width - xShift, zeroCoordinates.Y +
15));
                g.DrawString(p2.TotalTime.ToString(),
SystemFonts.DialogFont, Brushes.Black, new PointF(zeroCoordinates.X -
30, po2.Y));

            }

            private void pbGraph_Resize(object sender, EventArgs e)
            {
                zeroCoordinates = new Point(xShift,
pbGraph.ClientRectangle.Height - yShift);
            }

            private void panel1_Resize(object sender, EventArgs e)
            {
                pbGraph.Width = panel1.Width;
                pbGraph.Height = panel1.Height;

                if (Form1.PeriodTable[Form1.PeriodTable.Count -
1].TotalTime * verticalShift > pbGraph.Height)
                    pbGraph.Height =
(int)(Form1.PeriodTable[Form1.PeriodTable.Count - 1].TotalTime *
verticalShift) + 200;
            }
        }
    }

using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using PrefixManager.Classes;

namespace PrefixManager
{
    public partial class XmlGraphForm : Form
    {
        PointF zeroCoordinates;
        const int xShift = 50;
        const int yShift = 50;
        static readonly Color coordinatesColor = Color.Black;
        private float horizontalShift = 1;
        private float verticalShift = 5;
        Pen coordinatesPen = new Pen(coordinatesColor, 3f);
        Pen myPen = new Pen(Color.Red);
        Period p1, p2;
        Graphics g;
        Random rand = new Random();

        PointF po1 = new Point(), po2 = new Point();

        private List<Dictionary<int, Period>> files = new
List<Dictionary<int, Period>>();
        private List<Color> colors = new List<Color>();

        public XmlGraphForm()
        {
            InitializeComponent();

            coordinatesPen.EndCap = LineCap.ArrowAnchor;
            myPen.DashStyle = DashStyle.Dash;
        }

        private void pbGraph_Paint(object sender, PaintEventArgs e)
        {
            g = e.Graphics;
            g.SmoothingMode = SmoothingMode.AntiAlias;
            int index = -1;

            foreach (Dictionary<int, Period> dic in files)
            {
                index++;
                // if (pbGraph.ClientRectangle.Width >
Form1.table.Count)
                horizontalShift =
(float)((float)(pbGraph.ClientRectangle.Width - xShift -
zeroCoordinates.X) / dic.Count);


                //Draw Coordinates
                g.DrawLine(coordinatesPen, zeroCoordinates, new
PointF(pbGraph.ClientRectangle.Width - xShift, zeroCoordinates.Y));
```

```
                g.DrawLine(coordinatesPen, zeroCoordinates, new
PointF(zeroCoordinates.X, yShift));

                Pen p = new Pen(colors[index]);

                for (int i = 1; i < dic.Count; i++)
                {

                    p1 = dic[i];
                    p2 = dic[i + 1];


                    po1.X = i * horizontalShift + xShift;
                    po1.Y = (int)(pbGraph.ClientRectangle.Height -
yShift - p1.TotalTime * verticalShift);

                    po2.X = (i + 1) * horizontalShift + xShift;
                    po2.Y = (int)(pbGraph.ClientRectangle.Height -
yShift - p2.TotalTime * verticalShift);

                    g.DrawLine(p, po1, po2);
                }


                //Draw Red lines
                g.DrawLine(myPen, new PointF(zeroCoordinates.X,
po2.Y), po2);
                g.DrawLine(myPen, new PointF(po2.X,
zeroCoordinates.Y), po2);

                //Draw strings
                g.DrawString(dic.Count.ToString(),
SystemFonts.DialogFont, Brushes.Black, new
PointF(pbGraph.ClientRectangle.Width - xShift + 15, po2.Y));
                g.DrawString(p2.TotalTime.ToString(),
SystemFonts.DialogFont, Brushes.Black, new PointF(zeroCoordinates.X -
30, po2.Y));
            }
        }

        private void pbGraph_SizeChanged(object sender, EventArgs e)
        {
            zeroCoordinates = new Point(xShift,
pbGraph.ClientRectangle.Height - yShift);
        }

        private void addGraphXMLFileToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            string fileName = string.Empty;

            OpenFileDialog openDialog = new OpenFileDialog();
            openDialog.Filter = "XML Files|*.xml";

            if (openDialog.ShowDialog() == DialogResult.OK)
            {
                fileName = openDialog.FileName;
```

```
                Dictionary<int, Period> x =
XMLGraph.ReadFromFile(fileName);
                if (x != null)
                {
                    files.Add(x);

                    colors.Add(Color.FromArgb(rand.Next(255),
rand.Next(255), rand.Next(255)));
                    this.Refresh();
                }
            }
        }

        private void clearFilesToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            colors.Clear();
            foreach (Dictionary<int, Period> x in files)
                x.Clear();
            files.Clear();

            this.Refresh();
        }
    }
}
```

10–2– الشيفرة المصدرية لبرنامج المخدم بلغة الـ C# :

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Collections;
using System.Threading;
using System.Diagnostics;
using PrefixManager.Classes;
using System.Net;
using System.Net.Sockets;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
namespace PrefixManager
{
    public partial class ServerSide : Form
    {
        public static Dictionary<int, Period> PeriodTable = new
Dictionary<int, Period>();
        public static Dictionary<int, GeneralFactory> DfgTable;
        Thread thread;
        private DateTime startTime;
        private SchedulingAlgorithm algorithm;
        // New
        public dataSocket[] DataSoketList ;
        public int ClientNum ;
        public DateTime Total_Time;
        public int countTo;
        public int countFrom;
        public Socket m_mainSocket;
        public IPEndPoint ipLocal;
```

```
        public string IPstr = "";
        public string PortStr = "";
        public int PortInt = 0;
        public ArrayList m_workerSocketList =
ArrayList.Synchronized(new System.Collections.ArrayList());
        private int m_clientCount = 0;
        public AsyncCallback pfnWorkerCallBack;
        public delegate void UpdateClientListCallback();
        public delegate void UpdateRichEditCallback(string text);
        //
        public ServerSide()
        {
            InitializeComponent();
            PortStr = Port_Edit.Text;
            Control.CheckForIllegalCrossThreadCalls = false;
            IPstr = GetIP();
            IP_Edite.Text = IPstr;
            countTo = Convert.ToInt32( numTo.Value);
            countFrom = Convert.ToInt32(numFrom.Value);
        }
        private void Calc()
        {
            txtResults.Clear();
            listView1.Items.Clear();

            Period p;
            PeriodTable.Clear();
            int From = Convert.ToInt32(numFrom.Value);
            int To = Convert.ToInt32(numTo.Value);
            string temp2 = string.Empty, temp1 = string.Empty;
            int stepCount = To - From + 1;


            startTime = DateTime.Now;
            tsStartTime.Text = startTime.ToShortTimeString();
            lblElapsedTime.Text = string.Empty;

            Stopwatch w = new Stopwatch();
            DfgTable = new Dictionary<int, GeneralFactory>(To);

            tsProgressBar.Value = 0;

            for (int i = From; i <= To; i=i+50)
            {
                p = new Period();

                #region Infix Creation
                //Infix Creation
                InfixExpression infix = new InfixExpression(i);

                w.Start();
                infix.Create();
                w.Stop();

                p.InfixCreation = w.ElapsedMilliseconds;
                #endregion

                w.Reset();
```

```
                #region Postfix Creation
                //Postfix Creation
                PostfixExpression postfix = new
PostfixExpression(infix.InfixExp);
                w.Start();
                postfix.Create();
                temp1 = postfix.PostfixExp;
                w.Stop();

                p.PostfixCreation = w.ElapsedMilliseconds;

                #endregion

                w.Reset();

                #region DFG Creation
                //DFG Creation
                //SdfgFactory factory = new
SdfgFactory(postfix.PostfixExp, algorithm);
                GeneralFactory factory;
                if (rbDfg.Checked)
                    factory = new DfgFactory(postfix.PostfixExp);
                else
                    factory = new SdfgFactory(postfix.PostfixExp,
algorithm);
                w.Start();
                factory.Create();
                w.Stop();
                temp2 = factory.DfgExpression;
                DfgTable.Add(i, factory);

                p.DfgCreation = w.ElapsedMilliseconds;

                //DfgTable[i - 1] = factory;
                #endregion

                w.Reset();

                #region DFG Graph Creation
                //DFG Graph Creation
                DFGGraph g = new DFGGraph(factory.DfgList,
algorithm);
                w.Start();
                g.Draw();
                w.Stop();
                g.Dispose();

                p.DfgGraphCreation = w.ElapsedMilliseconds;
                #endregion

                w.Reset();
                infix.Dispose();

                //Draw Period


                PeriodTable[i] = p;
```

```
txtResults.AppendText(string.Format("Num:{0}\t\tInfix:{1}\tPostfix:{2
}\t\tDFG:{3}\tGraph:{4}\n",
                    i,
                    PeriodTable[i].InfixCreation,
                    PeriodTable[i].PostfixCreation,
                    PeriodTable[i].DfgCreation,
                    PeriodTable[i].DfgGraphCreation));

                if (i % 2000 == 0)
                {
                    XMLGraph.WriteToFile(PeriodTable,
string.Format("{0}.xml", i.ToString()));
                }

                //tsProgressBar.Value += stepCount / 100;
            }

            txtPostfixOutput.Text = temp1 + "\n";
            txtDFGOutput.Text = temp2 + "\n";
            timer1.Enabled = false;
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            if (PeriodTable != null)
                PeriodTable.Clear();
            if (DfgTable != null)
                DfgTable.Clear();
            txtResults.Clear();
            thread = new Thread(new ThreadStart(Calc));
            thread.Start();
        }

        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            if (thread != null)
            {
                thread.Abort();
            }
            CloseSockets();
        }

        private void closeToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Application.Exit();
        }

        private void graphComparerToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            XmlGraphForm form = new XmlGraphForm();
            form.Show();
        }

        private void btnPauses_Click(object sender, EventArgs e)
        {
```

```
            if (thread != null)
                thread.Suspend();
        }

        private void btnResume_Click(object sender, EventArgs e)
        {
            if (thread != null)
                thread.Resume();
        }

        private void generateTimeGraphToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            PostfixGraph gr = new PostfixGraph();
            gr.Show();
        }

        private void dFGGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            DFGGraph graph = new DFGGraph(DfgTable[DfgTable.Count -
1].DfgList, algorithm);
            graph.Show();
        }

        private void generateXMLToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                XMLGraph.WriteToFile(PeriodTable,
saveFileDialog1.FileName);
            }
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            TimeSpan t = DateTime.Now.Subtract(startTime);
            lblElapsedTime.Text = string.Format("{0}:{1}:{2}",
t.Hours, t.Minutes, t.Seconds);

        }

        private void rbSdfg_CheckedChanged(object sender, EventArgs
e)
        {
            if (rbSdfg.Checked == true)
            {
                pnlSdfg.Enabled = true;
                cbAlgorithm.SelectedIndex = 0;
            }
            else
                pnlSdfg.Enabled = false;
        }

        private void cbAlgorithm_SelectedIndexChanged(object sender,
EventArgs e)
        {
            algorithm =
```

```
(SchedulingAlgorithm)Enum.Parse(typeof(SchedulingAlgorithm),
                cbAlgorithm.SelectedItem.ToString());
        }

        private void ServerSide_Load(object sender, EventArgs e)
        {

        }

        private void StartBtn_Click(object sender, EventArgs e)
        {
            try
            {
                if (Port_Edit.Text == "")
                {
                    MessageBox.Show("Please enter a Port Number");
                    return;
                }
                StartBtn.Enabled = false;
                Stop_btn.Enabled = true;
                DistBtn.Enabled = true;
                // Bind to local IP Address...

                string portStr = PortStr;
                int port = System.Convert.ToInt32(portStr);
                // Create the listening socket...
                m_mainSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                IPEndPoint ipLocal = new IPEndPoint(IPAddress.Any,
port);
                // Bind to local IP Address...
                m_mainSocket.Bind(ipLocal);
                // Start listening...
                m_mainSocket.Listen(4);
                // Create the call back for any client connections...
                m_mainSocket.BeginAccept(new
AsyncCallback(OnClientConnect), null);

            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        // This method could be called by either the main thread or
any of the worker threads
        private void AppendToRichEditControl(string msg)
        {
            // Check to see if this method is called from a thread
            // other than the one created the control
            if (InvokeRequired)
            {
                // We cannot update the GUI on this thread.
                // All GUI controls are to be updated by the main
(GUI) thread.
                // Hence we will use the invoke method on the control
which will
                // be called when the Main thread is free
```

```
                // Do UI update on UI thread
                object[] pList = { msg };
                richTextBoxReceivedMsg.BeginInvoke(new
UpdateRichEditCallback(OnUpdateRichEdit), pList);
            }
            else
            {
                // This is the main thread which created this
control, hence update it
                // directly
                OnUpdateRichEdit(msg);
            }
        }
        private void OnUpdateRichEdit(string msg)
        {
            richTextBoxReceivedMsg.AppendText(msg);
        }
        public void OnClientConnect(IAsyncResult asyn)
        {
            try
            {
                // Here we complete/end the BeginAccept()
asynchronous call
                // by calling EndAccept() - which returns the
reference to
                // a new Socket object
                Socket workerSocket = m_mainSocket.EndAccept(asyn);

                // Now increment the client count for this client
                // in a thread safe manner
                Interlocked.Increment(ref m_clientCount);

                // Add the workerSocket reference to our ArrayList
                m_workerSocketList.Add(workerSocket);

                // Send a welcome message to client
                string msg = "Welcome client " + m_clientCount +
"\n";
                SendMsgToClient(msg, m_clientCount);

                // Update the list box showing the list of clients
(thread safe call)
                UpdateClientListControl();

                // Let the worker Socket do the further processing
for the
                // just connected client
                WaitForData(workerSocket, m_clientCount);

                // Since the main Socket is now free, it can go back
and wait for
                // other clients who are attempting to connect
                m_mainSocket.BeginAccept(new
AsyncCallback(OnClientConnect), null);
            }
            catch (ObjectDisposedException)
            {
                System.Diagnostics.Debugger.Log(0, "1", "\n
OnClientConnection: Socket has been closed\n");
```

```
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        // Start waiting for data from the client
        public void WaitForData(System.Net.Sockets.Socket soc, int
clientNumber)
        {
            try
            {
                if (pfnWorkerCallBack == null)
                {
                    // Specify the call back function which is to be
                    // invoked when there is any write activity by
the
                    // connected client
                    pfnWorkerCallBack = new
AsyncCallback(OnDataReceived);
                }
                SocketPacket theSocPkt = new SocketPacket(soc,
clientNumber);
                soc.BeginReceive(theSocPkt.dataBuffer, 0,
theSocPkt.dataBuffer.Length, SocketFlags.None, pfnWorkerCallBack,
theSocPkt);
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        // This the call back function which will be invoked when the
socket
        // detects any client writing of data on the stream
        public void OnDataReceived(IAsyncResult asyn)
        {
            SocketPacket socketData = (SocketPacket)asyn.AsyncState;
            int num = -1, p1 = -1, p2 = -1, p3 = -1, p4 = -1, p5 = -
1, i;
            string msg5="", msg6="",tem_Str="";
            try
            {
                // Complete the BeginReceive() asynchronous call by
EndReceive() method
                // which will return the number of characters written
to the stream
                // by the client
                int iRx =
socketData.m_currentSocket.EndReceive(asyn);
                char[] chars = new char[iRx + 1];
                // Extract the characters as a buffer
                System.Text.Decoder d =
System.Text.Encoding.UTF8.GetDecoder();
                int charLen = d.GetChars(socketData.dataBuffer, 0,
iRx, chars, 0);
                System.String szData = new System.String(chars);
                string msg = "" + socketData.m_clientNumber + ":";
                //AppendToRichEditControl(msg + szData);
```

```
if (szData[0] == '$')
{
    i = 1;
    while ((szData[i] != ':')&&(i < charLen))
    {
        tem_Str = tem_Str  + szData[i];
        i++;
    }
    i++;
    num = Convert.ToInt32(tem_Str);
    tem_Str = "";
    while ((szData[i] != ':') && (i < charLen))
    {
        tem_Str = tem_Str + szData[i];
        i++;
    }
    i++;
    p1 = Convert.ToInt32(tem_Str);
    tem_Str = "";
    while ((szData[i] != ':')&&(i < charLen))
    {
        tem_Str = tem_Str  + szData[i];
        i++;
    }
    i++;
    p2 = Convert.ToInt32(tem_Str);
    tem_Str = "";
    while ((szData[i] != ':')&&(i < charLen))
    {
        tem_Str = tem_Str + szData[i];
        i++;
    }
    i++;
    p3 = Convert.ToInt32(tem_Str);
    tem_Str = "";
    while ((szData[i] != ':')&&(i < charLen))
    {
        tem_Str = tem_Str + szData[i];
        i++;
    }
    i++;
    p4 = Convert.ToInt32(tem_Str);
    tem_Str = "";
    while ((szData[i] != ':') && (i < charLen))
    {
        tem_Str = tem_Str + szData[i];
        i++;
    }
    i++;
    p5 = Convert.ToInt32(tem_Str);
    int minusFactor = 0;
    for (int y = 0; y < socketData.m_clientNumber;
y++)
        if (m_workerSocketList[y] == null)
            minusFactor++;
    dataSocket tempDataSocket = new dataSocket();
    tempDataSocket.IsRecieved = true;
    tempDataSocket.infexCreation_Time = p1;
    tempDataSocket.PrefexCreation_Time = p2;
```

```
                    tempDataSocket.DFGCreation_Time = p3;
                    tempDataSocket.DFGCreation_Graph_Time = p4;
                    tempDataSocket.EndIndexInTree = p5;
                    tempDataSocket.PostfixOutput_Text = msg5;
                    tempDataSocket.DFGOutput_Text = msg6;
                    tempDataSocket.clientIndex =
socketData.m_clientNumber;
                    tempDataSocket.numNumber = num;



                    DataSoketList[socketData.m_clientNumber -
minusFactor - 1] = tempDataSocket;
                }
                isAllClientAnswer();
                // Send back the reply to the client
                //string replyMsg = "Server Reply:" +
szData.ToUpper();
                // Convert the reply to byte array
                //byte[] byData =
System.Text.Encoding.ASCII.GetBytes(replyMsg);

                //Socket workerSocket =
(Socket)socketData.m_currentSocket;
                //workerSocket.Send(byData);

                // Continue the waiting for data on the Socket
                WaitForData(socketData.m_currentSocket,
socketData.m_clientNumber);

            }
            catch (ObjectDisposedException)
            {
                System.Diagnostics.Debugger.Log(0, "1",
"\nOnDataReceived: Socket has been closed\n");
            }
            catch (SocketException se)
            {
                if (se.ErrorCode == 10054) // Error code for
Connection reset by peer
                {
                    string msg = "Client " +
socketData.m_clientNumber + " Disconnected" + "\n";
                    AppendToRichEditControl(msg);

                    // Remove the reference to the worker socket of
the closed client
                    // so that this object will get garbage collected
                    m_workerSocketList[socketData.m_clientNumber - 1]
= null;
                    UpdateClientListControl();
                }
                else
                {
                    MessageBox.Show(se.Message);
                }
            }
        }
        void SendMsgToClient(string msg, int clientNumber)
```

```
        {
            // Convert the reply to byte array
            byte[] byData = System.Text.Encoding.ASCII.GetBytes(msg);
            Socket workerSocket =
(Socket)m_workerSocketList[clientNumber - 1];
            workerSocket.Send(byData);
        }
        void UpdateClientListControl()
        {
            if (InvokeRequired)
            {
                ClientListBox.BeginInvoke(new
UpdateClientListCallback(UpdateClientList), null);
            }
            else
            {
                UpdateClientList();
            }
        }
        void UpdateClientList()
        {
            ClientNum = m_workerSocketList.Count;
            EdtClientNum.Text = ClientNum.ToString();
            ClientListBox.Items.Clear();
            for (int i = 0; i < m_workerSocketList.Count; i++)
            {
                string clientKey = Convert.ToString(i + 1);
                Socket workerSocket = (Socket)m_workerSocketList[i];
                if (workerSocket != null)
                {
                    if (workerSocket.Connected)
                    {
                        ClientListBox.Items.Add(clientKey);
                    }
                }
            }
        }
        void CloseSockets()
        {
            if (m_mainSocket != null)
            {
                m_mainSocket.Close();
            }
            Socket workerSocket = null;
            for (int i = 0; i < m_workerSocketList.Count; i++)
            {
                workerSocket = (Socket)m_workerSocketList[i];
                if (workerSocket != null)
                {
                    workerSocket.Close();
                    workerSocket = null;
                }
            }
        }

        private void Stop_btn_Click(object sender, EventArgs e)
        {
            CloseSockets();
            Stop_btn.Enabled = false;
```

```
            StartBtn.Enabled = true;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                string msg = richTextBoxSendMsg.Text;
                msg = "Server Msg: " + msg + "\n";
                byte[] byData =
System.Text.Encoding.ASCII.GetBytes(msg);
                Socket workerSocket = null;
                for (int i = 0; i < m_workerSocketList.Count; i++)
                {
                    workerSocket = (Socket)m_workerSocketList[i];
                    if (workerSocket != null)
                    {
                        if (workerSocket.Connected)
                        {
                            workerSocket.Send(byData);
                        }
                    }
                }
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            richTextBoxReceivedMsg.Clear();
        }
        String GetIP()
        {
            String strHostName = Dns.GetHostName();
            // Find host by name
            IPHostEntry iphostentry = Dns.GetHostByName(strHostName);
            // Grab the first IP addresses
            String IPStr = "";
            foreach (IPAddress ipaddress in iphostentry.AddressList)
            {
                IPStr = ipaddress.ToString();
                return IPStr;
            }
            return IPStr;
        }

        private void numTo_ValueChanged(object sender, EventArgs e)
        {
            NumtoDist.Value = numTo.Value;
            countTo = Convert.ToInt32( NumtoDist.Value);
        }

        private void NumtoDist_ValueChanged(object sender, EventArgs
e)
        {
            numTo.Value = NumtoDist.Value;
```

```
        }

        private void NumFromDist_ValueChanged(object sender,
EventArgs e)
        {
            countFrom = Convert.ToInt32(NumFromDist.Value);
            numFrom.Value = NumFromDist.Value;
        }

        private void numFrom_ValueChanged(object sender, EventArgs e)
        {
            countFrom = Convert.ToInt32(NumFromDist.Value);
            NumFromDist.Value = numFrom.Value;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                string msg = "Server CMD: Start Algorthim\n";
                byte[] byData =
System.Text.Encoding.ASCII.GetBytes(msg);
                Socket workerSocket = null;
                for (int i = 0; i < m_workerSocketList.Count; i++)
                {
                    workerSocket = (Socket)m_workerSocketList[i];
                    if (workerSocket != null)
                    {
                        if (workerSocket.Connected)
                        {
                            workerSocket.Send(byData);
                        }
                    }
                }
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        bool isAllClientAnswer()
        {
            for (int i = 0; i < ClientNum; i++)
            {
                if
((DataSoketList[i]==null)||(DataSoketList[i].IsRecieved == false))
                    return false;

            }
            return true;
        }
        private void button4_Click(object sender, EventArgs e)
        {
            try
            {

                if (DataSoketList != null)
                    DataSoketList = null;
                Socket workerSocket = null;
```

```
                    string msg = "";
                    ClientNum = 0;
                    for (int i = 0; i < m_workerSocketList.Count; i++)
                    {
                        workerSocket = (Socket)m_workerSocketList[i];
                        if (workerSocket != null)
                        {
                            ClientNum++;
                        }
                    }
                    DataSoketList = new dataSocket[ClientNum];
                    int xi =0;
                    if (ClientNum > 0)
                    {
                        DistBtn.Enabled = false;
                        dataSocket Temp_Data = new dataSocket();
                        DataSoketList[xi++] = Temp_Data;
                        int allCount = countTo - countFrom + 1;
                        int remainder = allCount % ClientNum;
                        int qutaValue = Convert.ToInt32((allCount -
remainder) / ClientNum);
                        byte[] byData = Encoding.ASCII.GetBytes(msg);
                        workerSocket = null;
                        for (int i = 0; i < m_workerSocketList.Count;
i++)
                        {
                            workerSocket = (Socket)m_workerSocketList[i];
                            if (workerSocket != null)
                            {
                                if (workerSocket.Connected)
                                {
                                    //workerSocket.Send(byData);
                                    if (remainder != 0)
                                    {
                                        msg = (qutaValue + 1).ToString();
                                        remainder--;
                                    }
                                    else
                                    {
                                        msg = qutaValue.ToString();
                                    }
                                    msg = msg + "\n";
                                    byData =
Encoding.ASCII.GetBytes(msg);

                                    workerSocket.Send(byData);
                                    msg = "Server CMD: Set Algorthim's

Values\n";
                                    byData =
Encoding.ASCII.GetBytes(msg);

                                    workerSocket.Send(byData);
                                }
                            }
                        }
                    }
                }
                catch (SocketException se)
                {
                    MessageBox.Show(se.Message);
                }
```

```
        }

        private void timer_DFG_Tick(object sender, EventArgs e)
        {
            if (isAllClientAnswer())
            {
                DistBtn.Enabled = true;
                timer_DFG.Stop();
            }
        }
        void StartClients(int CurrentValue)
        {
            try
            {
                Socket workerSocket = null;
                string msg = "";
                byte[] byData;
                ClientNum = 0;
                if (DataSoketList != null)
                    DataSoketList = null;
                for (int i = 0; i < m_workerSocketList.Count; i++)
                {
                    workerSocket = (Socket)m_workerSocketList[i];
                    if (workerSocket != null)
                    {
                        ClientNum++;
                    }
                }
                DataSoketList = new dataSocket[ClientNum];
                int xi = 0;
                if (ClientNum > 0)
                {
                    dataSocket Temp_Data = new dataSocket();
                    DataSoketList[xi++] = Temp_Data;
                    int remainder = CurrentValue % ClientNum;
                    int qutaValue = Convert.ToInt32((CurrentValue -
remainder) / ClientNum);
                    workerSocket = null;
                    for (int i = 0; i < m_workerSocketList.Count;
i++)
                    {
                        workerSocket = (Socket)m_workerSocketList[i];
                        if (workerSocket != null)
                        {
                            if (workerSocket.Connected)
                            {
                                if (remainder != 0)
                                {
                                    msg = (qutaValue + 1).ToString();
                                    remainder--;
                                }
                                else
                                {
                                    msg = qutaValue.ToString();
                                }
                                msg = msg + "\n";
                                byData =
Encoding.ASCII.GetBytes(msg);
                                workerSocket.Send(byData);
```

```
                                //msg = "Server CMD: Set Algorthim's
Values\n\0";
                                //byData =
Encoding.ASCII.GetBytes(msg);
                                //workerSocket.Send(byData);
                            }
                        }
                    }
                    msg = "Server CMD: Start Algorthim\n";
                    byData = Encoding.ASCII.GetBytes(msg);
                    for (int i = 0; i < m_workerSocketList.Count;
i++)
                    {
                        workerSocket = (Socket)m_workerSocketList[i];
                        if (workerSocket != null)
                        {
                            if (workerSocket.Connected)
                            {
                                workerSocket.Send(byData);
                            }
                        }
                    }
                }
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        void PrintClientResult()
        {
            dataSocket temp = new dataSocket();
            temp.clientIndex = 0;
            temp.numNumber = 0;
            temp.infexCreation_Time = 0;
            temp.PrefexCreation_Time = 0;
            temp.DFGCreation_Time = 0;
            temp.DFGCreation_Graph_Time = 0;
            for (int i = 0; i < ClientNum; i++)
            {
                temp.clientIndex++ ;
                temp.numNumber = temp.numNumber +
DataSoketList[i].numNumber;
                temp.infexCreation_Time = temp.infexCreation_Time +
DataSoketList[i].infexCreation_Time;
                temp.PrefexCreation_Time = temp.PrefexCreation_Time +
DataSoketList[i].PrefexCreation_Time;
                temp.DFGCreation_Time = temp.DFGCreation_Time +
DataSoketList[i].DFGCreation_Time;
                temp.DFGCreation_Graph_Time =
temp.DFGCreation_Graph_Time +
DataSoketList[i].DFGCreation_Graph_Time;
            }

txtResults.AppendText(string.Format("\t{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t
", temp.clientIndex, temp.numNumber, temp.infexCreation_Time,

temp.PrefexCreation_Time,temp.DFGCreation_Time,temp.DFGCreation_Graph
_Time));
```

```
            txtPostfixOutput.Clear();
            txtDFGOutput.Clear();
            if (ClientNum == 1)
            {
                txtPostfixOutput.AppendText("Client" +
DataSoketList[0].clientIndex.ToString());
                txtDFGOutput.AppendText("All the DFG Exsist in the
Client" + DataSoketList[0].clientIndex.ToString());
            }
            if (ClientNum == 2)
            {
                txtPostfixOutput.AppendText("Client" +
DataSoketList[0].clientIndex.ToString() + "Client" +
DataSoketList[1].clientIndex.ToString() + "+");

txtDFGOutput.AppendText(String.Format("Row1\t\t+\t\tClientNum{0}\t\tC
lientNum{1}\n",

DataSoketList[0].clientIndex.ToString(),

DataSoketList[1].clientIndex.ToString()));
            }
            if (ClientNum > 2)
            {
                txtPostfixOutput.AppendText("Client" +
DataSoketList[0].clientIndex.ToString() + "Client" +
DataSoketList[1].clientIndex.ToString() + "+");

txtDFGOutput.AppendText(String.Format("Row1\t\t+\t\tClientNum{0}\t\tC
lientNum{1}\n",

DataSoketList[0].clientIndex.ToString(),

DataSoketList[1].clientIndex.ToString()));
                for (int i = 2; i < ClientNum; i++)
                {
                    txtPostfixOutput.AppendText("Client" +
DataSoketList[i].clientIndex.ToString() + "+");

txtDFGOutput.AppendText(String.Format("Row{0}\t\t+\t\tRow{1}\t\t\tCli
entNum{2}\n", i, i - 1, DataSoketList[i].clientIndex.ToString()));
                }
            }



        }
        private void FullStartBtn_Click(object sender, EventArgs e)
        {
            DateTime currentTime ;
            int T1;
            try
            {
                DistBtn.Enabled = false;
                StartDist.Enabled = false;
                txtResults.Clear();
                txtPostfixOutput.Clear();
                txtDFGOutput.Clear();
```

```
                txtResults.AppendText(string.Format("        Client
Count: Number:   Infix:    Postfix:       DFG:       Graph:
Total:      \n"));
                for (int i = Convert.ToInt32(numFrom.Value) ; i <=
numTo.Value; i++)
                {
                    Total_Time = DateTime.Now;
                    //T1 = Total_Time.t
                    StartClients(i);
                    while (!(isAllClientAnswer()))
                    {
                        timer_DFG.Start();

                    }
                    PrintClientResult();
                    currentTime = DateTime.Now;
                    TimeSpan ff;
                    ff = currentTime - Total_Time;
                    T1 = ff.Milliseconds;
                    txtResults.AppendText(string.Format(T1.ToString()
+ "\t\n"));
                }
                DistBtn.Enabled = true;
                StartDist.Enabled = true;
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }

        }
    }

    public class SocketPacket
    {
        public System.Net.Sockets.Socket m_currentSocket;
        public int m_clientNumber;
        // Buffer to store the data sent by the client
        public byte[] dataBuffer = new byte[1024];
        // Constructor which takes a Socket and a client number
        public SocketPacket(System.Net.Sockets.Socket socket, int
clientNumber)
        {
            m_currentSocket = socket;
            m_clientNumber = clientNumber;
        }
    }
    public class dataSocket
    {
        public int clientIndex;
        public int numNumber;
        public int infexCreation_Time;
        public int PrefexCreation_Time;
        public int DFGCreation_Time;
        public int DFGCreation_Graph_Time;
        public int EndIndexInTree;
        public string PostfixOutput_Text;
        public string DFGOutput_Text;
        public bool IsRecieved;
```

```
        public dataSocket()
        {
            numNumber = 0;
            EndIndexInTree = 0;
            clientIndex = -1;
            IsRecieved = false;
            PostfixOutput_Text = "";
            DFGOutput_Text = "";
        }
    }
}
```

10-3- الشيفرة المصدرية لبرنامج الزبون بلغة الـ C# :

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Threading;
using System.Diagnostics;
using PrefixManager.Classes;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace PrefixManager
{
    public partial class ClientSide : Form
    {
        public static Dictionary<int, Period> PeriodTable = new
Dictionary<int, Period>();
        public static Dictionary<int, GeneralFactory> DfgTable;
        Thread thread;
        private DateTime startTime;
        private SchedulingAlgorithm algorithm;
        // New Client Data
        String StartCommand   = "Server CMD: Start Algorthim\n\0";
        String CounterCommand = "Server CMD: Set Algorthim's
Values\n\0";
        String PrevMSG = "";
        IAsyncResult m_result;
        byte[] m_dataBuffer = new byte[10];
        public Socket m_clientSocket;
        public AsyncCallback m_pfnCallBack;
        // End
        public ClientSide()
        {
            InitializeComponent();
            textBoxIP.Text = GetIP();
            Control.CheckForIllegalCrossThreadCalls = false;
        }
        //------------------------------------------------------
        // This is a helper function used (for convenience) to
        // get the IP address of the local machine
        //------------------------------------------------------
        String GetIP()
        {
```

```
            String strHostName = Dns.GetHostName();
            // Find host by name
            IPHostEntry iphostentry = Dns.GetHostByName(strHostName);
            // Grab the first IP addresses
            String IPStr = "";
            foreach (IPAddress ipaddress in iphostentry.AddressList)
            {
                IPStr = ipaddress.ToString();
                return IPStr;
            }
            return IPStr;
        }
        private void Calc()
        {
            txtResults.Clear();
            listView1.Items.Clear();
            Period p;
            PeriodTable.Clear();
            int From = Convert.ToInt32(numFrom.Value);
            int To = Convert.ToInt32(numTo.Value);
            p = new Period();
            if (From > 0)
            {
                string temp2 = string.Empty, temp1 = string.Empty;
                int stepCount = To - From + 1;
                startTime = DateTime.Now;
                tsStartTime.Text = startTime.ToShortTimeString();
                lblElapsedTime.Text = string.Empty;
                Stopwatch w = new Stopwatch();
                DfgTable = new Dictionary<int, GeneralFactory>(To);
                tsProgressBar.Value = 0;
                int i = From;
                #region Infix Creation
                //Infix Creation
                InfixExpression infix = new InfixExpression(i);
                w.Start();
                infix.Create();
                w.Stop();
                p.InfixCreation = w.ElapsedMilliseconds;
                #endregion
                w.Reset();
                #region Postfix Creation
                //Postfix Creation
                PostfixExpression postfix = new
PostfixExpression(infix.InfixExp);
                w.Start();
                postfix.Create();
                temp1 = postfix.PostfixExp;
                w.Stop();
                p.PostfixCreation = w.ElapsedMilliseconds;
                #endregion
                w.Reset();
                #region DFG Creation
                //DFG Creation
                //SdfgFactory factory = new
SdfgFactory(postfix.PostfixExp, algorithm);
                GeneralFactory factory;
                if (rbDfg.Checked)
                    factory = new DfgFactory(postfix.PostfixExp);
```

```
                else
                    factory = new SdfgFactory(postfix.PostfixExp,
algorithm);
                w.Start();
                factory.Create();
                w.Stop();
                temp2 = factory.DfgExpression;
                DfgTable.Add(i, factory);
                p.DfgCreation = w.ElapsedMilliseconds;
                //DfgTable[i - 1] = factory;
                #endregion
                w.Reset();
                #region DFG Graph Creation
                //DFG Graph Creation
                DFGGraph g = new DFGGraph(factory.DfgList,
algorithm);
                w.Start();
                g.Draw();
                w.Stop();
                g.Dispose();
                p.DfgGraphCreation = w.ElapsedMilliseconds;
                #endregion
                w.Reset();
                infix.Dispose();
                PeriodTable[i] = p;

txtResults.AppendText(string.Format("Num:{0}\t\tInfix:{1}\tPostfix:{2
}\t\tDFG:{3}\tGraph:{4}\n",
                                                    i,

PeriodTable[i].InfixCreation,

PeriodTable[i].PostfixCreation,

PeriodTable[i].DfgCreation,

PeriodTable[i].DfgGraphCreation));
                // ----- XMLGraph.WriteToFile(PeriodTable,
string.Format("{0}.xml", i.ToString()));
                //tsProgressBar.Value += stepCount / 100;
                txtPostfixOutput.Text = temp1 + "\n";
                txtDFGOutput.Text = temp2 + "\n";
                SentEndClinetWork(p);
                timer1.Enabled = false;
            }
            else
            {
                p.InfixCreation = 0;
                p.PostfixCreation = 0;
                p.DfgCreation = 0;
                p.DfgGraphCreation = 0;

txtResults.AppendText(string.Format("Num:{0}\t\tInfix:{1}\tPostfix:{2
}\t\tDFG:{3}\tGraph:{4}\n",0,0,0,0,0));
                // ----- XMLGraph.WriteToFile(PeriodTable,
string.Format("{0}.xml", i.ToString()));
                //tsProgressBar.Value += stepCount / 100;
                txtPostfixOutput.Text = " \n";
                txtDFGOutput.Text =  " \n";
```

```
                SentEndClinetWork(p);
                timer1.Enabled = false;
            }
        }
        void StartFunc()
        {
            if (PeriodTable != null)
                PeriodTable.Clear();
            if (DfgTable != null)
                DfgTable.Clear();
            thread = new Thread(new ThreadStart(Calc));
            thread.Start();
        }
        private void btnStart_Click(object sender, EventArgs e)
        {
            StartFunc();
        }
        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            if (thread != null)
                thread.Abort();
            if (m_clientSocket != null)
            {
                m_clientSocket.Close();
                m_clientSocket = null;
            }
        }
        public void WaitForData()
        {
            try
            {
                if (m_pfnCallBack == null)
                {
                    m_pfnCallBack = new
AsyncCallback(OnDataReceived);
                }
                SocketPacket theSocPkt = new SocketPacket();
                theSocPkt.thisSocket = m_clientSocket;
                // Start listening to the data asynchronously
                m_result =
m_clientSocket.BeginReceive(theSocPkt.dataBuffer, 0,
theSocPkt.dataBuffer.Length, SocketFlags.None, m_pfnCallBack,
theSocPkt);
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        public void OnDataReceived(IAsyncResult asyn)
        {
            try
            {
                SocketPacket theSockId  =
(SocketPacket)asyn.AsyncState;
                int iRx                 =
theSockId.thisSocket.EndReceive(asyn);
                char[] chars            = new char[iRx + 1];
```

```
                Decoder d                = Encoding.UTF8.GetDecoder();
                int charLen              =
d.GetChars(theSockId.dataBuffer, 0, iRx, chars, 0);
                String szData            = new String(chars);
                richTextRxMessage.Text   = richTextRxMessage.Text +
szData;
                if (szData == StartCommand)
                {
                    numTo.Value = Convert.ToDecimal(PrevMSG);
                    numFrom.Value = numTo.Value;
                    StartFunc();
                }
                PrevMSG = szData;
                WaitForData();
            }
            catch (ObjectDisposedException)
            {
                System.Diagnostics.Debugger.Log(0, "1",
"\nOnDataReceived: Socket has been closed\n");
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        private void closeToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Application.Exit();
        }
        private void graphComparerToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            XmlGraphForm form = new XmlGraphForm();

            form.Show();
        }
        private void btnPauses_Click(object sender, EventArgs e)
        {
            if (thread != null)
                thread.Suspend();
        }
        private void btnResume_Click(object sender, EventArgs e)
        {
            if (thread != null)
                thread.Resume();
        }
        private void generateTimeGraphToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            PostfixGraph gr = new PostfixGraph();
            gr.Show();
        }
        private void dFGGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            DFGGraph graph = new DFGGraph(DfgTable[DfgTable.Count -
1].DfgList, algorithm);
            graph.Show();
```

```
        }
        private void generateXMLToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                XMLGraph.WriteToFile(PeriodTable,
saveFileDialog1.FileName);
            }
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            TimeSpan t = DateTime.Now.Subtract(startTime);
            lblElapsedTime.Text = string.Format("{0}:{1}:{2}",
t.Hours, t.Minutes, t.Seconds);
        }
        private void rbSdfg_CheckedChanged(object sender, EventArgs
e)
        {
            if (rbSdfg.Checked == true)
            {
                pnlSdfg.Enabled = true;
                cbAlgorithm.SelectedIndex = 0;
            }
            else
                pnlSdfg.Enabled = false;
        }
        private void cbAlgorithm_SelectedIndexChanged(object sender,
EventArgs e)
        {
            algorithm =
(SchedulingAlgorithm)Enum.Parse(typeof(SchedulingAlgorithm),cbAlgorit
hm.SelectedItem.ToString());
        }
        private void buttonConnect_Click(object sender, EventArgs e)
        {
            // See if we have text on the IP and Port text fields
            if (textBoxIP.Text == "" || textBoxPort.Text == "")
            {
                MessageBox.Show("IP Address and Port Number are
required to connect to the Server\n");
                return;
            }
            try
            {
                UpdateControls(false);
                // Create the socket instance
                m_clientSocket = new
Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

                // Cet the remote IP address
                IPAddress ip = IPAddress.Parse(textBoxIP.Text);
                int iPortNo =
System.Convert.ToInt16(textBoxPort.Text);
                // Create the end point
                IPEndPoint ipEnd = new IPEndPoint(ip, iPortNo);
                // Connect to the remote host
                m_clientSocket.Connect(ipEnd);
```

```
                    if (m_clientSocket.Connected)
                    {

                        UpdateControls(true);
                        //Wait for data asynchronously
                        WaitForData();
                    }
                }
                catch (SocketException se)
                {
                    string str;
                    str = "\nConnection failed, is the server running?\n"
+ se.Message;
                    MessageBox.Show(str);
                    UpdateControls(false);
                }
            }
            private void buttonDisconnect_Click(object sender, EventArgs
e)
            {
                if (m_clientSocket != null)
                {
                    m_clientSocket.Close();
                    m_clientSocket = null;
                    UpdateControls(false);
                }
            }
            private void ButtonSendMessage_Click(object sender, EventArgs
e)
            {
                try
                {
                    string msg = richTextTxMessage.Text;
                    // New code to send strings
                    NetworkStream networkStream = new
NetworkStream(m_clientSocket);
                    System.IO.StreamWriter streamWriter = new
System.IO.StreamWriter(networkStream);
                    streamWriter.WriteLine(msg);
                    streamWriter.Flush();
                }
                catch (SocketException se)
                {
                    MessageBox.Show(se.Message);
                }
            }
            private void SentEndClinetWork( Period Peri)
            {
                try
                {
                    string msg = "Client Result:End Work Signal";
                    msg = "$";
                    msg = msg + numFrom.Value.ToString();
                    msg = msg + ":";
                    msg = msg + Peri.InfixCreation.ToString();
                    msg = msg + ":";
                    msg = msg + Peri.PostfixCreation.ToString();
                    msg = msg + ":";
                    msg = msg + Peri.DfgCreation.ToString();
```

```
                msg = msg + ":";
                msg = msg + Peri.DfgGraphCreation.ToString();
                msg = msg + ":";
                int cc =  Convert.ToInt32( numFrom.Value);
                if (cc > 0)
                    msg = msg + Convert.ToString(cc * 2 - 1);
                else
                    msg = msg + 0;
                /*msg = msg + ":";
                msg = msg + txtPostfixOutput.Text;
                msg = msg + ":";
                msg = msg + txtDFGOutput.Text;*/
                //if (m_clientSocket != null)
                    NetworkStream networkStream = new
NetworkStream(m_clientSocket);
                System.IO.StreamWriter streamWriter = new
System.IO.StreamWriter(networkStream);
                streamWriter.WriteLine(msg);
                streamWriter.Flush();
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        private void UpdateControls(bool connected)
        {
            buttonConnect.Enabled = !connected;
            buttonDisconnect.Enabled = connected;
            numFrom.Enabled =!connected;
            numTo.Enabled = !connected;
            btnStart.Enabled = !connected;
            btnResume.Enabled = !connected;
            btnPauses.Enabled = !connected;
            string connectStatus = connected ? "Connected" : "Not
Connected";
            textBoxConnectStatus.Text = connectStatus;
        }
        private void btnClear_Click(object sender, EventArgs e)
        {
            richTextTxMessage.Clear();
        }
    }
    public class SocketPacket
    {
        public System.Net.Sockets.Socket thisSocket;
        public byte[] dataBuffer = new byte[1024];
    }
}
```

المراجع

[1] Nadine Azemard; Lars Svensson , 2007-Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation , 99,104,299.

[2] VOLNEI A.P., 2004- Circuit Design with VHDL.

[3] CAMPOSANO R., 1996- Behavioral synthesis. 33 rd Design Automation Conference, 33-34.

[4] COMAP, Consortium for Mathematics and Its Applications (U.S.), 2002- For all practical purposes: mathematical literacy in today's world,150-200.

[5] Duen-Jeng Wang , 1995- 5Multiprocessor Synthesis of Embedded Real-time DSP, 384 .

[6] ERIK L.; JIM C., 2007- Interactive Digital Filter Design- Online Tool for IIR Filter and FIR Filter Design.

[7] Julius O. Smith III, 2000- Introduction to Digital Filters with Audio Applications.

[8] M. D. Lutovac; D. V. Tosic; B. L. Evans, 1999- Filter Design for Signal Processing using MATLAB and *Mathematica*.

[9] Rahman Jamal; Mike Cerna; John Hanks, Designing Filters Using the Digital Filter Design Toolkit.

[10] L.J Hafer and A.C. Parker, Aformal Method for the Specification, Analysis, and Design of RTL Digital Logic, IEEE Trans. Computer-Aided Design.

[11] NATIONAL INSTRUMENTS("NI")., 2006- Designing Filters Using the NI Labview Digital Filter Design Toolkit, 150.

[12] FLOTTES M.L.; HAMMAD D., 1994- automatic Synthesis of bisted data paths from high level specification. European Design and Test conference, 591-598.

[13] Daniel D. Gajski, 1997- Principles of digital design,2,26,64,377.

[14] Zhe Ma; Pol Marchal; Daniele Paolo Scarpazza; Peng, 2007-Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms, 60-70.

[15] http//ccrma.standford.edu/.

[16] BERREBI E., 1996- Combined control flow and data flow dominated High-level synthesis. 33 rd Design Automation Conference. 573-578.

[17] Daniel D. Gajski, 1997- Principles of digital design, 2,26,64,377.

[18] John P. Elliott, 1999- Understanding behavioral synthesis: a practical guide to high-level design, 95,311,269.

[19] Michael Pinedo, 2008- Scheduling: Theory, Algorithms, and Systems.

[20] AGRAWAL V.D.; CHENG K.T., 1990- Finite State Machine Synthesis with Embedded Test Function. *Journal of Electronic Testing: Theory and Applications,* **1**, 221-228.

[21] Dinesh P.Mehta; Sataj Sahni, 2004- Handbook of Data Structures and Applications, 250-300.

[22] Michael T. Goodrich; Roberto Tamassia, 2005- Data Structures and Algorithms in Java.

[23] David R. Richardson, 2002- The Book on Data Structures.

[24] Adam Drozdek, 2000- Data Structures and Algorithm in C++.

[25] Kruse R. L. ,1984- Data Structures and Program Design.

[26] David Houde; Timothy Hoffman, 2001- TCP/IP for Windows 2000, 160-200.

[27] Charles M. Kozierok, 2005- The TCP/IP guide: a comprehensive, illustrated Internet protocols reference.

[28] IEEE Computer Society, 1997- Advances in parallel and distributed computing, 19-21.

[29] David B. Makofske; Michael J. Donahoo; Kenneth L. Calvert, 2004- TCP/IP sockets in C#: practical guide for programmers.

[30] Sukumar Ghosh, 2006- Distributed systems: an algorithmic approach.

[31] Andrew S. Tanenbaum; Maarten van Steen, 2006- Distributed Systems: Principles and Paradigms, 50-150.